2

Robot Manipulators and Control Systems

2.1 Introduction

This book focuses on industrial robotic manipulators and on industrial manufacturing cells built using that type of robots. This chapter covers the current practical methodologies for kinematics and dynamics modeling and computations. The kinematics model represents the motion of the robot without considering the forces that cause the motion. The dynamics model establishes the relationships between the motion and the forces involved, taking into account the masses and moments of inertia, i.e., the dynamics model considers the masses and inertias involved and relates the forces with the observed motion, or instead calculates the forces necessary to produce the required motion. These topics are considered very important to study and efficient use of industrial robots.

Both the kinematics and dynamics models are used currently to design, simulate, and control industrial robots. The kinematics model is a prerequisite for the dynamics model and fundamental for practical aspects like motion planning, singularity and workspace analysis, and manufacturing cell graphical simulation. For example, the majority of the robot manufacturers and many independent software vendors offer graphical environments where users, namely developers and system integrators, can design and simulate their own manufacturing cell projects (Figure 2.1).

Kinematics and dynamics modeling is the subject of numerous publications and textbooks [1-4]. The objective here is to present the topics without prerequisites, covering the fundamentals. Consequently, a real industrial robot will be used as an example which makes the chapter more practical, and easier to read. Nevertheless, the reader is invited to seek further explanation in the following very good sources:

1. *Introduction to Robotics*, JJ Craig, John Willey and Sons, Chapters 2 to 7.

2. *Modeling and Control of Robotic Manipulators*, F. Sciavicco and B. Siciliano, Mcgraw Hill, Chapters 2 to 5.
3. *Handbook of Industrial Robotics*, 2nd edition, Shimon Nof, Chapter 6 written by A. Goldenberg and M. Emani.

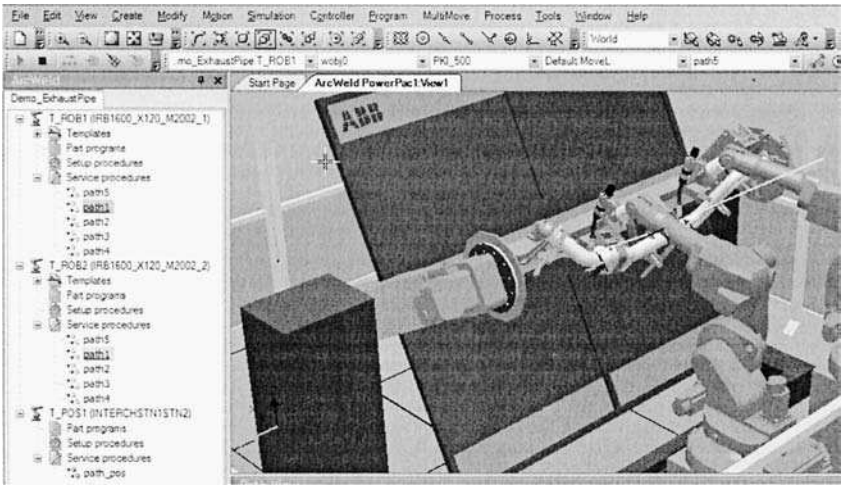


Figure 2.1 Aspect of a graphical simulation package (*RobotStudio* – ABB Robotics)

Another important practical aspect is the way how these topics are implemented and used by actual robot control systems. This chapter also reviews the fundamental aspects of robot control systems from the perspective of an engineer and of a system integrator. The objective is to introduce the main components and modules of modern robot control systems, by examining some of the control systems available commercially.

2.2 Kinematics

Actual industrial robot manipulators are very advanced machines exhibiting high precision and repeatability. It's common to have medium payload robots (16 to 20kg of payload) offering repeatability up to 0.1 mm, with smaller robots exhibiting even better performances (up to 0.01 mm). These industrial robots are basically composed by rigid links, connected in series by joints (normally six joints), having one end fixed (base) and another free to move and perform useful work when properly tooled (*end-effector*). As with the human arm, robot manipulators use the first three joints (arm) to position the structure and the remaining joints (wrist, composed of three joints in the case of the industrial manipulators) are used to orient the *end-effector*. There are five types of arms commonly used by actual industrial robot manipulators (Figure 2.2): *cartesian*, *cylindrical*, *polar*, *SCARA* and *revolution*.

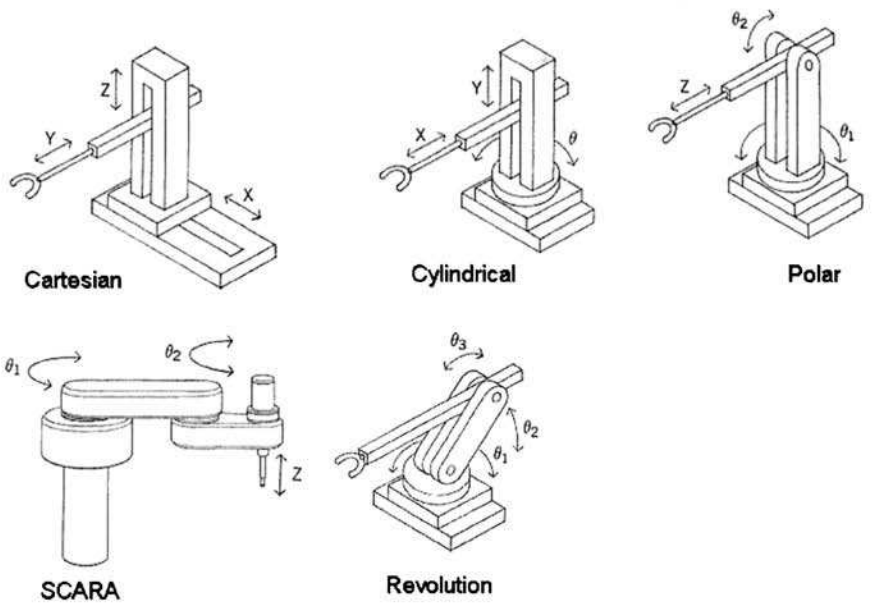


Figure 2.2 Types of arms used with actual robot manipulators

In terms of wrist designs, there are two main configurations (Figure 2.3):

1. pitch-yaw-roll (XYZ) like the human arm
2. roll-pitch-roll (ZYZ) or spherical wrist

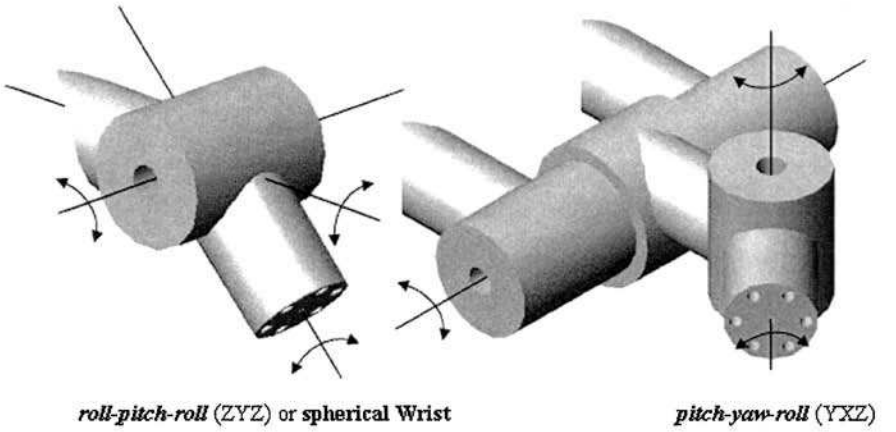


Figure 2.3 Wrist design configurations

The spherical wrist is the most popular because it is mechanically simpler to implement. Nevertheless, it exhibits singular configurations that can be identified

and consequently avoided when operating with the robot. The trade between simplicity of robust solutions and the existence of singular configurations is favorable to the spherical wrist design, and that is the reason for its success.

The position and orientation of the robot's *end-effector* (tool) is not directly measured but instead computed using the individual joint position readings and the kinematics of the robot. Inverse kinematics is used to obtain the joint positions required for the desired *end-effector* position and orientation [1]. Those transformations involve three different representation spaces: a*ctuator space*, j*oint space* and c*artesian space*. The relationships between those spaces will be established here, with application to an ABB IRB1400 industrial robot (Figure 2.4). The discussion will be kept general for an anthropomorphic[1] manipulator with a spherical wrist[2].

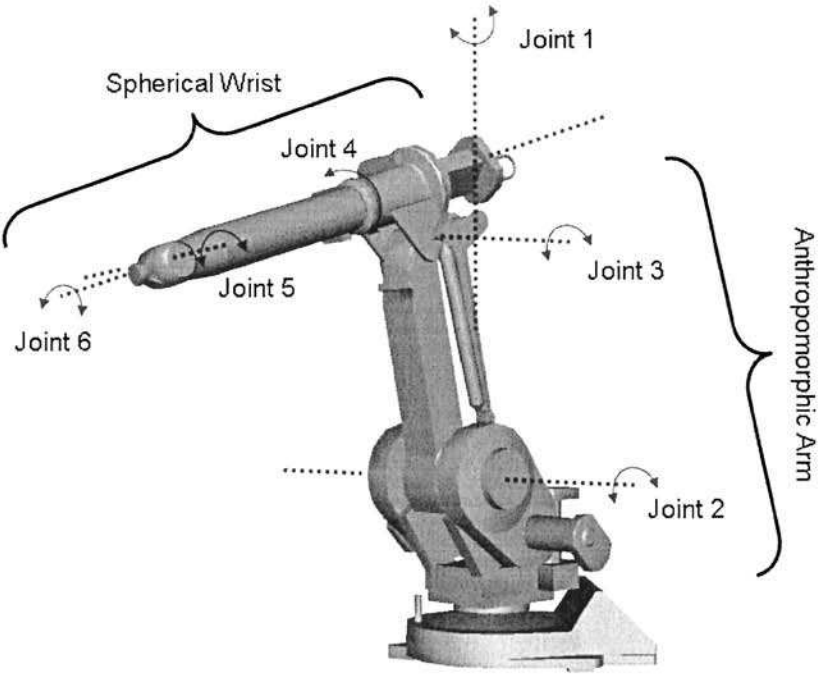


Figure 2.4 ABB IRB1400 industrial robot

[1] An anthropomorphic structure is a set of three revolute joints, with the first joint orthogonal to the other two which are parallel

[2] A spherical wrist has three revolute joints whose axes intersect at a single point

Table 2.1 *Denavit-Hartenberg* parameters for the IRB1400

Link	θ_i (°)	α_{i-1} (°)	a_{i-1} (mm)	d_i (mm)
1	θ_1 (0°)	0°	0	475
2	θ_2 (90°)	90°	150	0
3	θ_3 (0°)	0°	600	0
4	θ_4 (0°)	90°	120	720
5	θ_5 (0°)	-90°	0	0
6	θ_6 (0°)	90°	0	85 + d

where d is an extra length associated with the *end-effector*

Table 2.2 Workspace and maximum velocities for the IRB1400

Joint	Workspace (°)	Maximum Velocity (°/s)
1	+170° to -170°	110°/s
2	+70° to -70°	110°/s
3	+70° to -65°	110°/s
4	+150° to -150°	280°/s
5	+115° to -115°	280°/s
6	+300° to -300°	280°/s

Figure 2.5 represents, for simplicity, the robot manipulator axis lines and the assigned frames. The *Denavit-Hartenberg* parameters, the joint range and velocity limits are presented in Tables 2.1 and 2.2. The represented frames and associated parameters were found using Craig's convention [1].

2.2.1 Direct Kinematics

By simple inspection of Figure 2.5 it is easy to conclude that the last three axes form a set of *ZYZ Euler* angles [1,2] with respect to frame 4. In fact, the overall rotation produced by those axes is obtained from:

1. rotation about Z_4 by θ_4
2. rotation about Y'_4=Z'_5 by θ_5
3. rotation about Z''_4=Z''_6 by θ_6.[3]

which gives the following rotation matrix,

[3] Y'_4 corresponds to axis Y_4 after rotation about Z_4 by θ_4 and Z''_4 corresponds to Z_4 after rotation about Y'_4=Z'_5 by θ_5

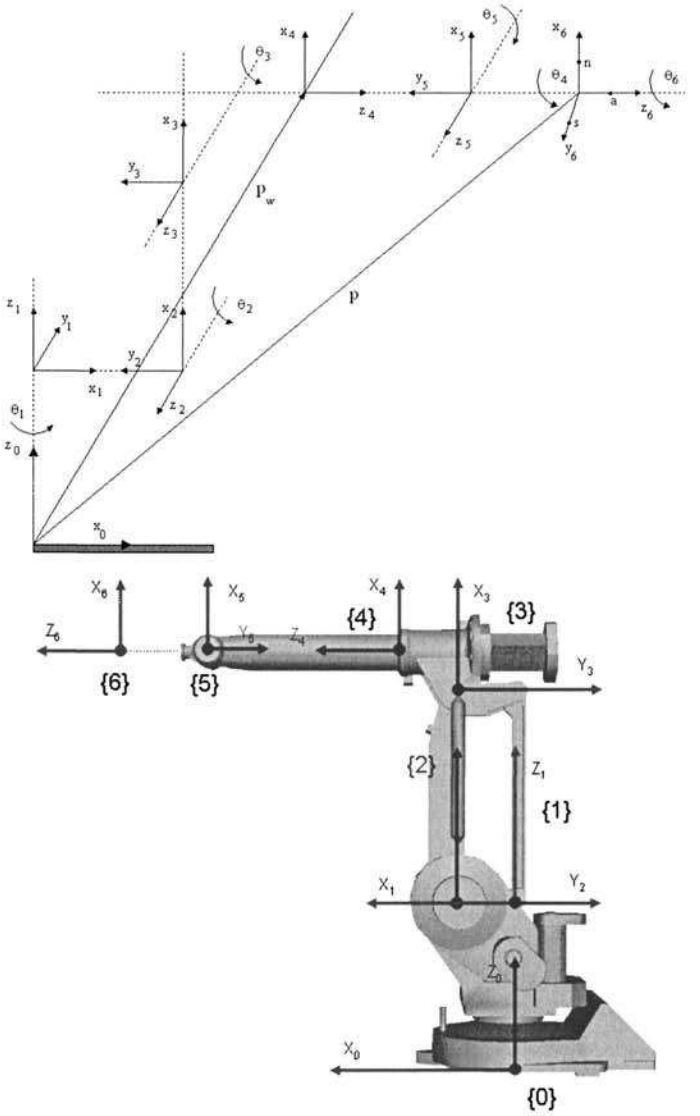


Figure 2.5 Link frame assignment

$$R_{Euler} = R_z(\theta_4).R_{y'4}(\theta_5).R_{z''4'}(\theta_6) =$$

$$= \begin{bmatrix} c_4 & -s_4 & 0 \\ s_4 & c_4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & 0 & s_5 \\ 0 & 1 & 0 \\ -s_5 & 0 & c_5 \end{bmatrix} \begin{bmatrix} c_6 & -s_6 & 0 \\ s_6 & c_6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$= \begin{bmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_5 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 \\ -s_5c_6 & s_5s_6 & c_5 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = R$$

The above rotation matrix R, in accordance with the assigned frame settings, should verify the following two equations:

$$R_6^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} .R$$

$$R(\theta_4 = 0) = R_6^4 \tag{2.2}$$

The values of θ_4, θ_5 and θ_6 can be now obtained. Comparing r_{13} with r_{23} (considering $s_5 \neq 0$) results in,

$$\theta_4 = A\tan 2(r_{23}, r_{13}) \tag{2.3}$$

Squaring and summing r_{13} and r_{23} and comparing the result with r_{33} gives,

$$\theta_5 = A\tan 2(\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \tag{2.4}$$

if a positive square-root of $r_{13}^2 + r_{23}^2$ is chosen: this assumption limits the range of θ_5 to $[0,\pi]$.

Using the same argument now considering elements r_{31} and r_{32} the following is obtained for θ_6:

$$\theta_6 = A\tan 2(r_{32}, -r_{31}) \tag{2.5}$$

For $\theta_5 \in [-\pi,0]$ the solution is:

$$\begin{aligned} \theta_4 &= A\tan 2(-r_{23}, -r_{13}) \\ \theta_5 &= A\tan 2(-\sqrt{r_{13}^2 + r_{23}^2}, r_{33}) \\ \theta_6 &= A\tan 2(-r_{32}, r_{31}) \end{aligned} \tag{2.6}$$

The IRB1400 is an anthropomorphic manipulator with spherical wrist. The anthropomorphic structure of the first three joints is the one that offers better

dexterity to the robot manipulator. The first three joints are used to position the wrist. The orientation of the wrist is managed by the wrist spherical structure, which is also the one that gives higher dexterity. Using the link transformation matrix definition derived at [1],

$$T_i^{i-1} = \begin{bmatrix} c_i & -s_i & 0 & a_{i-1} \\ s_i c\alpha_{i-1} & c_i c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} d_i \\ s_i s\alpha_{i-1} & c_i s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

the direct kinematics of the ABB IRB1400 robot manipulator can be easily obtained (as presented in Figure 2.6).

$$T_1^0 = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2^1 = \begin{bmatrix} -s_2 & -c_2 & 0 & a_1 \\ 0 & 0 & -1 & 0 \\ c_2 & -s_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & a_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^3 = \begin{bmatrix} c_4 & -s_4 & 0 & a_3 \\ 0 & 0 & -1 & -d_4 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_5^4 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_5 & -c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^5 = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & -1 & -d_6 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2^0 = \begin{bmatrix} -c_1 s_2 & -c_1 c_2 & s_1 & a_1 c_1 \\ -s_1 s_2 & -s_1 c_2 & -c_1 & a_1 s_1 \\ c_2 & -s_2 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^0 = \begin{bmatrix} -c_1 s_{23} & -c_1 c_{23} & s_1 & -a_2 c_1 s_2 + a_1 c_1 \\ -s_1 s_{23} & -s_1 c_{23} & -c_1 & -a_2 s_1 s_2 + a_1 s_1 \\ c_{23} & -s_{23} & 0 & a_2 c_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^0 = \begin{bmatrix} -c_1 s_{23} c_4 + s_1 s_4 & c_1 s_{23} s_4 + s_1 c_4 & c_1 c_{23} & d_4 c_1 c_{23} - a_3 c_1 s_{23} - a_2 c_1 s_2 + a_1 c_1 \\ -s_1 s_{23} c_4 - c_1 s_4 & s_1 s_{23} s_4 - c_1 c_4 & s_1 c_{23} & d_4 s_1 c_{23} - a_3 s_1 s_{23} - a_2 s_1 s_2 + a_1 s_1 \\ c_{23} c_4 & c_{23} s_4 & s_{23} & d_4 s_{23} + a_3 c_{23} + a_2 c_2 + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^3 = \begin{bmatrix} c_4 c_5 c_6 - s_4 s_6 & -c_4 c_5 s_6 - s_4 c_6 & c_4 s_5 & d_6 c_4 s_5 + a_3 \\ s_5 c_6 & -s_5 s_6 & -c_5 & -d_6 c_5 - d_4 \\ s_4 c_5 c_6 + c_4 s_6 & -s_4 c_5 s_6 + c_4 c_6 & s_4 s_5 & d_6 s_4 s_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_6^4 = \begin{bmatrix} c_5c_6 & -c_5s_6 & s_5 & d_6s_5 \\ s_6 & c_6 & 0 & 0 \\ -s_5c_6 & s_5s_6 & c_5 & d_6c_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } T_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x^0 \\ r_{21} & r_{22} & r_{23} & p_y^0 \\ r_{31} & r_{32} & r_{33} & p_z^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ with,}$$

$r_{11} = ((s_1s_4 - c_1s_{23}c_4)c_5 - c_1c_{23}s_5)c_6 + (c_1s_{23}s_4 + s_1c_4)s_6$
$r_{12} = ((-s_1s_4 + c_1s_{23}c_4)c_5 + c_1c_{23}s_5)s_6 + (c_1s_{23}s_4 + s_1c_4)c_6$
$r_{13} = (-c_1s_{23}c_4 + s_1s_4)s_5 + c_1c_{23}c_5$
$r_{21} = ((-s_1s_{23}c_4 - c_1s_4)c_5 - s_1c_{23}s_5)c_6 + (s_1s_{23}s_4 - c_1c_4)s_6$
$r_{22} = ((s_1s_{23}c_4 + c_1s_4)c_5 + s_1c_{23}s_5)s_6 + (s_1s_{23}s_4 - c_1c_4)c_6$
$r_{23} = (-s_1s_{23}c_4 - c_1s_4)s_5 + s_1c_{23}c_5$
$r_{31} = (c_{23}c_4c_5 - s_{23}s_5)c_6 - c_{23}s_4s_6$
$r_{32} = (-c_{23}c_4c_5 + s_{23}s_5)s_6 - c_{23}s_4c_6$
$r_{33} = c_{23}c_4s_5 + s_{23}c_5$
$p^0_x = ((-c_1s_{23}c_4 + s_1s_4)s_5 + c_1c_{23}c_5)d_6 + d_4c_1c_{23} - a_3c_1s_{23} - a_2c_1s_2 + a_1c_1$
$p^0_y = ((-s_1s_{23}c_4 - c_1s_4)s_5 + s_1c_{23}c_5)d_6 + d_4s_1c_{23} - a_3s_1s_{23} - a_2s_1s_2 + a_1s_1$
$p^0_z = d_6(c_{23}c_4s_5 + s_{23}c_5) + d_4s_{23} + a_3c_{23} + a_2c_2 + d_1$

Figure 2.6 Direct kinematics of an ABB IRB 1400 industrial robot

Having derived the direct kinematics of the IRB 1400, it's now possible to obtain the *end-effector* position and orientation from the individual joint angles $(\theta_1,\theta_2,\theta_3,\theta_4;\theta_5,\theta_6)$.

2.2.2 Inverse Kinematics

Inverse kinematics deals with the problem of finding the required joint angles to produce a certain desired position and orientation of the *end-effector*. Finding the inverse kinematics solution for a general manipulator can be a very tricky task. Generally they are non-linear equations. Close-form solutions may not be possible and multiple, infinity, or impossible solutions can arise. Nevertheless, special cases have a closed-form solution and can be solved.

The sufficient condition for solving a six-axis manipulator is that it must have three consecutive revolute axes that intersect at a common point: *Pieper* condition [5]. Three consecutive revolute parallel axes is a special case of the above condition, since parallel lines can be considered to intersect at infinity. The ABB IRB 1400 meets the *Pieper* condition due to the spherical wrist.

For these types of manipulators, i.e. manipulators that meet the *Pieper* condition, it is possible to decouple the inverse kinematics problem into two sub-problems: position and orientation. A simple strategy [1,2] can then be used to solve the inverse kinematics, by separating the position problem from the orientation problem. Consider Figure 2.5, where the position and orientation of the *end-*

effector is defined in terms of **p** and $R_6^0 = [n \quad s \quad a]$. The wrist position ($\mathbf{p}_w$) can be found using

$$\mathbf{p}_w = \mathbf{p} - d_6.\mathbf{a} \tag{2.8}$$

It is now possible to find the inverse kinematics for θ_1,θ_2 and θ_3 and solve the first inverse kinematics sub-problem, i.e, the position sub-problem. Considering Figure 2.7 it is easy to see that

$$\theta_1 = A\tan 2(p_{wy}, p_{wx})^{4} \tag{2.9}$$

Once θ_1 is known the problem reduces to solving a planar structure. Looking to Figure 2.7 it is possible to successively write

$$p_{wx1} = \sqrt{p_{wx}^2 + p_{wy}^2} \tag{2.10}$$

$$p_{wz1} = p_{wz} - d_1 \tag{2.11}$$

$$p_{wx1'} = p_{wx1} - a_1 \tag{2.12}$$

$$p_{wy1'} = p_{wy1} \tag{2.13}$$

$$p_{wz1'} = p_{wz1} \tag{2.14}$$

and

$$p_{wx1'} = -a_2 s_2 + a_x c_{23'} \tag{2.15}$$

$$p_{wz1'} = a_2 c_2 + a_x s_{23'} \tag{2.16}$$

---

[4] Another possibility would be $\theta_1 = \pi + A\tan 2(p_{wy}, p_{wx})$ if we set $\theta_2 \rightarrow \pi - \theta_2$

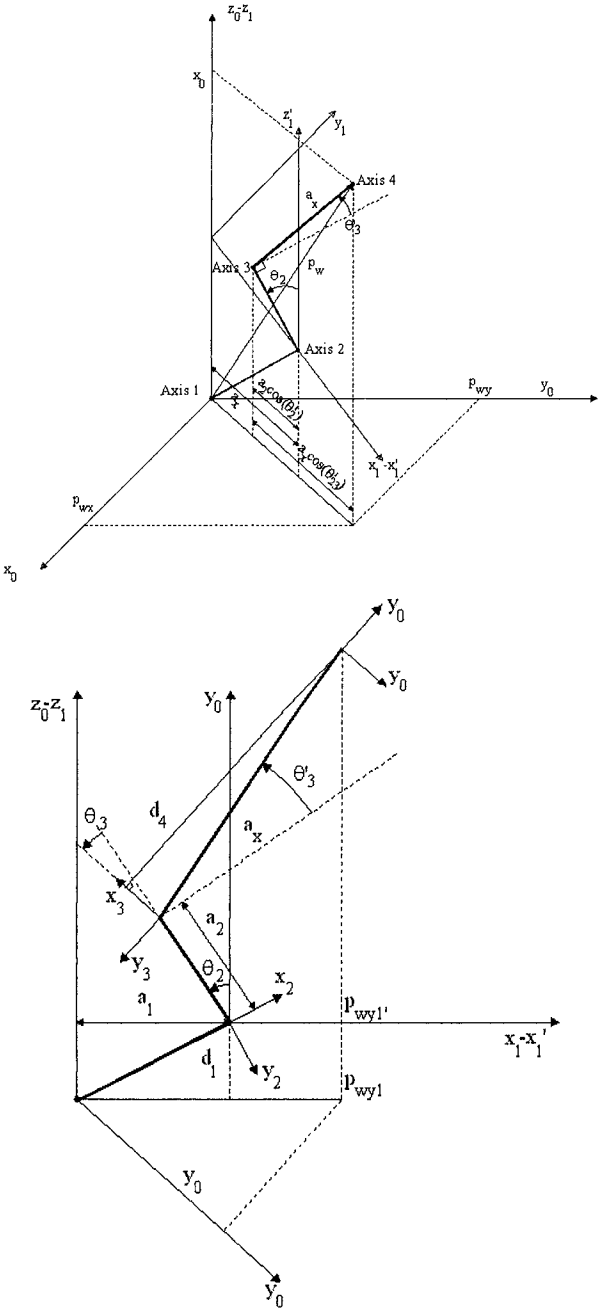


Figure 2.7 Anthropomorphic structure

Squaring and summing equations (2.15) and (2.16) results in

$$p_{wx1'}^2 + p_{wz1'}^2 = a_2^2 + a_x^2 + a_2 a_x s_{3'} \tag{2.17}$$

which gives

$$s_{3'} = \frac{p_{wx1'}^2 + p_{wz1'}^2 - a_2^2 - a_x^2}{2a_2 a_x} \tag{2.18}$$

Setting $c_{3'} = \pm\sqrt{1 - s_{3'}^2}$ the solution for θ'_3 will be

$$\begin{aligned} \theta'_3 &= A\tan 2(s_{3'}, c_{3'}) \\ \theta_3 &= \theta'_3 - A\tan(a_3 / d_4) \end{aligned} \tag{2.19}$$

Now, using θ'_3 in (2.15)-(2.16) results in a system with two equations with s_2 and c_2 unknowns:

$$\begin{aligned} p_{wx1'} &= a_2 c_2 + a_x (c_2 c_{3'} - s_2 s_{3'}) \\ p_{wz1'} &= a_2 s_2 + a_x (s_2 c_{3'} + s_{3'} c_2) \end{aligned} \tag{2.20}$$

Solving for s_2 and c_2 gives

$$s_2 = \frac{-(a_2 + a_x s_{3'}) p_{wx1'} + a_x c_{3'} p_{wz1'}}{a_2^2 + a_x^2 + 2a_2 a_x s_{3'}} \tag{2.21}$$

$$c_2 = \frac{(a_2 + a_x s_{3'}) p_{wz1'} + a_x c_{3'} p_{wx1'}}{a_2^2 + a_x^2 + 2a_2 a_x s_{3'}} \tag{2.22}$$

and the solution for θ_2 will be

$$\theta_2 = A\tan 2(s_2, c_2) \tag{2.23}$$

To solve the second inverse kinematics sub-problem (orientation), i.e., to find the required joint angles θ_4, θ_5 and θ_6 corresponding to a given *end-effector* orientation R_6^3, we simply take advantage of the special configuration of the last three joints. Because the orientation of the *end-effector* is defined by R_6^0, it's simple to get R_6^3 from,

$$R_6^3 = (R_3^0)^{-1} . R_6^0 = (R_3^0)^T . R_6^0 \tag{2.24}$$

which gives

$$R_6^3 = \begin{bmatrix} -c_1 s_{23} & -s_1 s_{23} & c_{23} \\ -c_1 c_{32} & -s_1 c_{23} & -s_{23} \\ s_1 & -c_1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.25}$$

with

$$\begin{aligned} r_{11} &= -c_1 s_{23} a_{11} - s_1 s_{23} a_{21} + c_{23} a_{31} & r_{12} &= -c_1 s_{23} a_{12} - s_1 s_{23} a_{22} + c_{23} a_{32} \\ r_{13} &= -c_1 s_{23} a_{13} - s_1 s_{23} a_{23} + c_{23} a_{33} & r_{23} &= -c_1 c_{23} a_{13} - s_1 c_{23} a_{23} - s_{23} a_{33} \\ r_{33} &= s_1 a_{13} - c_1 a_{23} & & \\ r_{21} &= -c_1 c_{23} a_{11} - s_1 c_{23} a_{21} - s_{23} a_{31} & r_{22} &= -c_1 c_{23} a_{12} - s_1 c_{23} a_{22} - s_{23} a_{32} \\ r_{31} &= s_1 a_{11} - c_1 a_{21} & r_{32} &= s_1 a_{12} - c_1 a_{22} \end{aligned}$$

It is now possible to use the previous result for the ZYZ Euler angles to obtain the solutions for θ_4, θ_5 and θ_6 .

For $\theta_5 \in [0, \pi]$ the solution is

$$\begin{aligned} \theta_4 &= A\tan 2(r_{33}, r_{13}) \\ \theta_5 &= A\tan 2(\sqrt{r_{13}^2 + r_{33}^2}, -r_{23}) \\ \theta_6 &= A\tan 2(-r_{22}, r_{21}) \end{aligned} \tag{2.26}$$

For $\theta_5 \in [-\pi,0]$ the solution is

$$\begin{aligned} \theta_4 &= A\tan 2(-r_{33}, -r_{13}) \\ \theta_5 &= A\tan 2(-\sqrt{r_{13}^2 + r_{33}^2}, r_{23}) \\ \theta_6 &= A\tan 2(r_{22}, -r_{21}) \end{aligned} \tag{2.27}$$

2.3 Jacobian

In this section, the equations necessary to compute the jacobian of the ABB IRB1400 industrial robot are presented and the jacobian is obtained. Nevertheless, the discussion will be kept general for an anthropomorphic robot manipulator. In the process, the equations that describe the linear and angular velocities, static forces, and moments of each of the manipulator links are also presented and the corresponding developments applied to the selected robot.

The jacobian of any robot manipulator structure is a matrix that relates the *end-effector* linear and angular Cartesian velocities with the individual joint velocities:

$$V = \begin{bmatrix} v \\ w \end{bmatrix} = J(\theta).\dot{\theta} \tag{2.28}$$

where $J(\theta)$ is the jacobian matrix of the robot manipulator, $\dot{\theta} = [\dot{\theta}_1, \dot{\theta}_2, ..., \dot{\theta}_n]^T$ is the joint velocity vector, $v = [v_1, v_2, v_3]^T$ is the *end-effector* linear velocity vector, and $w = [w_1, w_2, w_3]^T$ is the *end-effector* angular velocity vector.

The jacobian is an n×m matrix, where n is the number of degrees of freedom of the robot manipulator and m is the number of joints. Considering an anthropomorphic robot manipulator with a spherical wrist, the corresponding jacobian will be a 6×6 matrix. Basically there are two ways to compute the jacobian:

1. By direct differentiation of the direct kinematics function with respect to the joint variables. This usually leads to the so-called *analytical jacobian*,

$$\dot{x} = \begin{bmatrix} \dot{p} \\ \dot{\phi} \end{bmatrix} = J_A(\theta).\dot{\theta} \tag{2.29}$$

   where $\dot{p}$ is the time derivative of the position of the *end-effector* frame with respect to the base frame, $\dot{\phi}$ is the time derivative of the orientation vector expressed in terms of three variables (for instance, ZYZ Euler angles). Obviously, $\dot{p}$ is the translational velocity of the *end-effector* and $\dot{\phi}$ is the rotational velocity.

2. By computing the contributions of each joint velocity to the components of the *end-effector* Cartesian linear and angular velocities. This procedure leads to the *geometric jacobian*.

Generally, the analytical and geometrical jacobian are different from each other. Nevertheless, it is always possible to write

$$w = T(\phi).\dot{\phi} \tag{2.30}$$

where T is a transformation matrix from $\dot{\phi}$ to w. Once $T(\phi)$ is given, the analytical jacobian and geometric jacobian can be related by

$$V = \begin{bmatrix} I & 0 \\ 0 & T(\phi) \end{bmatrix}.\dot{x} = T_J(\phi).\dot{x} \tag{2.31}$$

which gives

$$J = T_J(\phi).J_A \tag{2.32}$$

Here the geometric jacobian will be calculated, because in the process the linear and angular velocities of each link will also be obtained. Nevertheless, the analytical jacobian should be used when the variables are defined in the operational space.

First the equations for the link linear and angular velocities and accelerations [1,2] will be obtained. Associating a frame to each rigid body, the rigid body motion can be described by the relative motion of the associated frames. Consider a frame {B} associated with a point D (Figure 2.8).

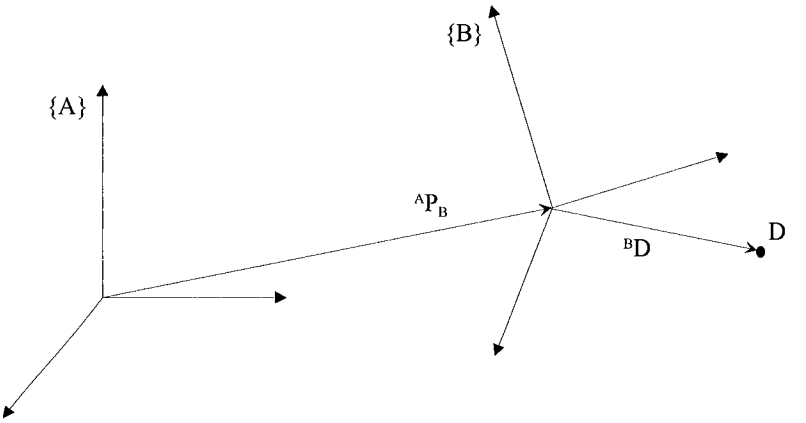


Figure 2.8 Describing point D relative to a stationary frame

The position vector of point D in frame {B} is ^{B}D and the relative velocity of D described about an arbitrary stationary frame {A} is [6],

$$^{A}V_{D} = {}^{A}V_{B} + {}^{A}_{B}R \; {}^{B}V_{D} \tag{2.33}$$

If the relative motion between {A} and {B} is non-linear then (2.33) is not valid. The relative motion between two frames {A} and {B} has generally two components: a linear component $^{A}V_{B}$ and a non-linear component (the angular or rotational acceleration) $^{A}\Omega_{B}$ as in (Figure 2.9).

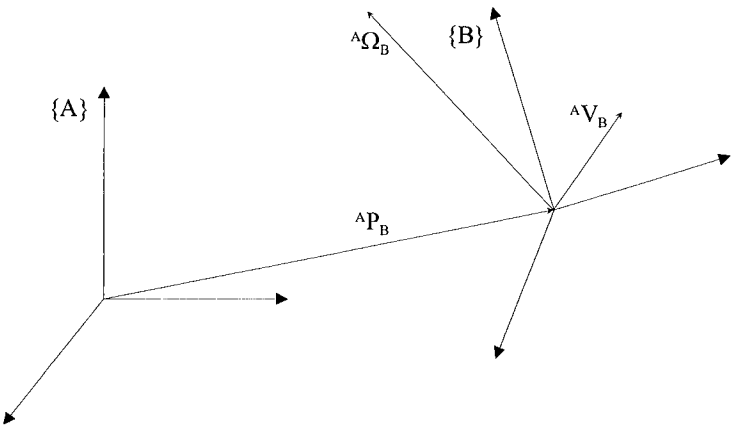


Figure 2.9 Relative motion between two frames {A} and {B}

In that general case it can be written [1,6,7],

$$^{A}V_{D} = {}^{A}V_{B} + {}^{A}_{B}R\ {}^{B}V_{D} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}D \tag{2.34}$$

where $^{A}V_{D}$ is the linear velocity of the origin of frame {B} about frame {A}, $^{A}_{B}R\ ^{B}V_{D}$ is the linear velocity of point D about frame {B} expressed in terms of {A} (i.e., $^{A}_{B}R\ ^{B}V_{D} = {}^{A}(^{B}V_{D})$), $^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}D = {}^{A}\Omega_{B} \times {}^{A}D$ is the linear velocity of point D about {A} expressed in terms of {A} as the result of the angular velocity $^{A}\Omega_{B}$ of {B} about {A}.

If D is stationary in {B} ($^{B}V_{D} = 0$) and the origins of {A} and {B} are coincident, i.e., the relative motion of D about {A} is only due to the rotation motion of {B} about {A} described by $^{A}\Omega_{B}$, then $^{A}V_{D} = {}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}D$. This equation can also be obtained by differentiation of

$$^{A}D = {}^{A}_{B}R\ {}^{B}D \tag{2.35}$$

which yields

$$^{A}\dot{D} = {}^{A}_{B}\dot{R}\ {}^{B}D + {}^{A}_{B}R\ {}^{B}\dot{D} \tag{2.36}$$

or since in this special case $^{A}_{B}R\ ^{B}\dot{D} = 0$,

$$^{A}V_{D} = {}^{A}_{B}\dot{R}\ {}^{B}D \tag{2.37}$$

Substituting in (2.37) ${}^{B}D = {}^{A}_{B}R^{-1}\ {}^{A}D$ results in

$$ {}^{A}V_D = {}^{A}_{B}\dot{R}\ {}^{A}_{B}R^{-1}\ {}^{A}D \tag{2.38} $$

Because ${}^{A}_{B}R$ is an orthonormal matrix, we can write [1,7],

$$ {}^{A}_{B}\dot{R}\ {}^{A}_{B}R^{-1} = {}^{A}_{B}S \tag{2.39} $$

where ${}^{A}_{B}S$ is a *skew-symmetric* matrix associated with ${}^{A}_{B}R$.

Using (2.39) in (2.38) gives

$$ {}^{A}V_D = {}^{A}_{B}S\ {}^{A}D \tag{2.40} $$

The skew-symmetric matrix ${}^{A}_{B}S$ defined in (2.39) is called *angular velocity matrix.*

Writing S as

$$ S = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} \tag{2.41} $$

and the vector Ω (3×1) as

$$ \Omega = \begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{2.42} $$

results in

$$ S\,D = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix} . \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} = \begin{bmatrix} -\Omega_z D_y + \Omega_y D_z \\ \Omega_z D_x - \Omega_x D_z \\ -\Omega_y D_x + \Omega_x D_y \end{bmatrix} = \Omega \times D \tag{2.43} $$

where $D = (D_x, D_y, D_z)^T$ is a position vector. The vector Ω associated with the angular velocity matrix is called an *angular velocity vector.* Using (2.43) and (2.40) gives

$$ {}^{A}V_D = {}^{A}\Omega_B \times {}^{A}D \tag{2.44} $$

Considering now the linear and angular accelerations of each link, it's possible to write by direct differentiation of (2.34),

$$ {}^{A}\dot{V}_{D} = {}^{A}\dot{V}_{B} + ({}^{A}_{B}R\ {}^{B}V_{D})' + {}^{A}\dot{\Omega}_{B} \times {}^{A}_{B}R\ {}^{B}D + {}^{A}\Omega_{B} \times ({}^{A}_{B}R\ {}^{B}D)' \tag{2.45} $$

or since,

$$ ({}^{A}_{B}R\ {}^{B}V_{D})' = {}^{A}_{B}R\ {}^{B}\dot{V}_{D} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}V_{D} $$

and

$$ ({}^{A}_{B}R\ {}^{B}D)' = {}^{A}_{B}R\ {}^{B}V_{D} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}D, $$

$$ \begin{aligned} {}^{A}\dot{V}_{D} = {}^{A}\dot{V}_{B} + {}^{A}_{B}R\ {}^{B}\dot{V}_{D} + 2\,{}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}V_{D} + \\ + {}^{A}\dot{\Omega}_{B} \times {}^{A}_{B}R\ {}^{B}D + {}^{A}\Omega_{B} \times ({}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}D) \end{aligned} \tag{2.46} $$

The above equation is the general equation for the linear acceleration of point D about {A} and expressed in terms of {A}. If ${}^{B}D$ is a constant vector (like in robotics applications) then equation (2.46) simplifies to

$$ {}^{A}\dot{V}_{D} = {}^{A}\dot{V}_{B} + {}^{A}\dot{\Omega}_{B} \times {}^{A}_{B}R\ {}^{B}D + {}^{A}\Omega_{B} \times ({}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}D) \tag{2.47} $$

because ${}^{B}V_{D} = {}^{B}\dot{V}_{D} = 0$.

If we consider a third frame {C}, with ${}^{A}\Omega_{B}$ being the angular velocity of {B} about {A} and ${}^{B}\Omega_{C}$ the angular velocity of {B} about {C}, then the angular velocity of {C} about {A} is,

$$ {}^{A}\Omega_{C} = {}^{A}\Omega_{B} + {}^{A}_{B}R\ {}^{B}\Omega_{C} \tag{2.48} $$

Taking the derivative of (2.48) results in

$$ {}^{A}\dot{\Omega}_{C} = {}^{A}\dot{\Omega}_{B} + ({}^{A}_{B}R\ {}^{B}\Omega_{C})' = {}^{A}\dot{\Omega}_{B} + {}^{A}_{B}R\ {}^{B}\dot{\Omega}_{C} + {}^{A}\Omega_{B} \times {}^{A}_{B}R\ {}^{B}\Omega_{C} \tag{2.49} $$

This is a very useful equation to compute the angular acceleration propagation from link to link.

Let's apply this to a robot manipulator. As mentioned before we will consider only rigid manipulators with revolutionary joints, with the base frame as the reference frame.

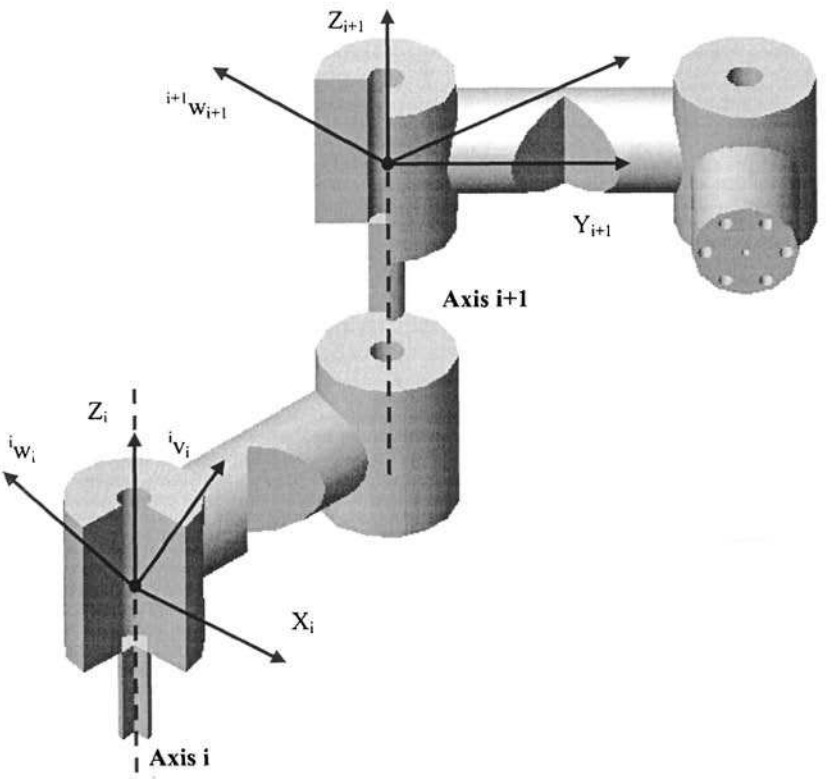


Figure 2.10 Linear and angular velocity vectors of adjacent links

The angular velocity of link (i+1), expressed in terms of {i}, is given by[5]

$$^iw_{i+1} = {}^iw_i + {}^i_{i+1}R\ \dot{\theta}_{i+1}\ {}^{i+1}Z_{i+1} \tag{2.50}$$

It is equal to the angular velocity of link (i) plus the angular velocity of joint (i+1) about Z_{i+1} expressed in terms of {i}.

Multiplying both sides of (2.50) by ${}^{i+1}_iR$ results in the angular velocity of link (i+1) expressed in terms of {i+1},

$$^{i+1}w_{i+1} = {}^{i+1}_iR\ {}^iw_{i+1} = {}^{i+1}_iR\ {}^iw_i + \dot{\theta}_{i+1}\ {}^{i+1}Z_{i+1} \tag{2.51}$$

---

[5] Note that wi+1 = 0Ωi+1 and that iwi+1 is the same quantity expressed in terms of {i}.

The linear velocity of the origin of {i+1}, expressed in terms of {i}, is given by

$$^{i}v_{i+1} = {}^{i}v_i + {}^{i}w_i \times {}^{i}P_{i+1} \tag{2.52}$$

It is equal to the linear velocity of the origin of {i} plus a term that results from the rotation of the link (i+1) about Z_{i+1}. The same solution can be obtained from (7) by making $^{i}P_{i+1}$ constant in {i}, i.e., by making $^{i}v_{i+1} = 0$.

Multiplying both sides of (2.52) by $^{i+1}_{i}R$ we get the linear velocity of link (i+1) expressed in terms of {i+1}

$$^{i+1}v_{i+1} = {}^{i+1}_{i}R\ ({}^{i}v_i + {}^{i}w_i \times {}^{i}P_{i+1}) \tag{2.53}$$

Applying (2.51) and (2.53) from link to link, the equations for $^{n}w_n$ and $^{n}v_n$ (where **n** is the number of joints) will be obtained. The equations for $^{0}w_n$ and $^{0}v_n$ can be obtained by pre-multiplying $^{n}w_n$ and $^{n}v_n$ by $^{0}_{n}R$:

$$^{0}w_n = {}^{0}_{n}R\ {}^{n}w_n \tag{2.54}$$

$$^{0}v_n = {}^{0}_{n}R\ {}^{n}v_n \tag{2.55}$$

It's also important to know how forces and moments distribute through the links and joints of the robot manipulator in a static situation, i.e., how to compute the forces and moments that keep the manipulator still in the various operating static configurations. Considering the manipulator at some configuration, the static equilibrium is obtained by proper balancing of the forces and moments applied to each joint and link, i.e., by cancelling the resultant of all the forces applied to the center of mass of each link (static equilibrium). The objective is to find the set of moments that should be applied to each joint to keep the manipulator in static equilibrium for some working configuration (Figure 2.11).

Considering,

f_i = force applied at link (i) by link (i-1)
n_i = moment in link (i) due to link (i-1)

the static equilibrium is obtained when

$$^{i}f_i - {}^{i}f_{i+1} = 0 \quad \text{and} \quad {}^{i}n_i - {}^{i}n_{i+1} - {}^{i}P_{i+1} \times {}^{i}f_{i+1} = 0 \tag{2.56}$$

i.e., when,

$$^{i}f_i = {}^{i}f_{i+1} \tag{2.57}$$

and

$$^{i}n_i = {}^{i}n_{i+1} + {}^{i}P_{i+1} \times {}^{i}f_{i+1} \tag{2.58}$$

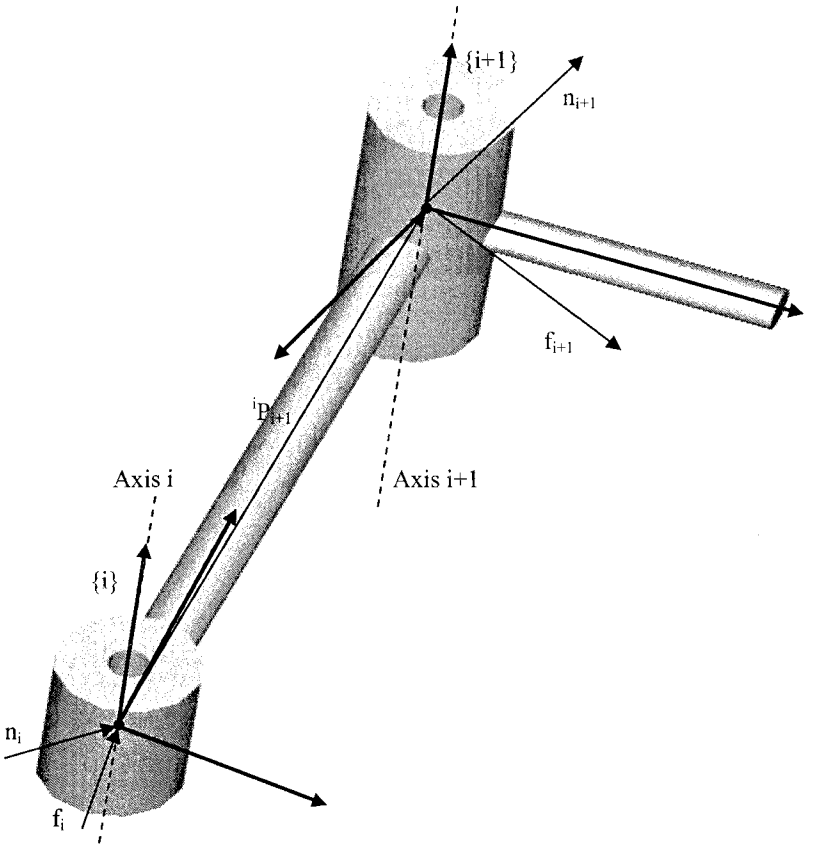


Figure 2.11 Static equilibrium: force balancing over link (i)

Writing the above equations in their own reference frame gives

$$^{i}f_i = {}^{i}_{i+1}R \; {}^{i+1}f_{i+1} \tag{2.59}$$

$$^{i}n_i = {}^{i}_{i+1}R \; {}^{i+1}n_{i+1} + {}^{i}P_{i+1} \times {}^{i}f_i \tag{2.60}$$

To compute the set of joint moments that hold the manipulator in static equilibrium we must obtain, for each joint (i), the projection of $^{i}n_i$ over the joint axis

$$\tau_i = {}^{i}n_i^{T} \, {}^{i}Z_i \tag{2.61}$$

Returning to the jacobian, from (2.54)-(2.55) it's possible to write

$$^0w_{i+1} = {}^0w_i + {}^i_{i+1}R\ (\dot{\theta}_{i+1}\ {}^{i+1}Z_{i+1}) \tag{2.62}$$

$$^0v_{i+1} = {}^0v_i + {}^0w_i \times {}^0P^i_{i+1} \tag{2.63}$$

Using (1) and (2.62)-(2.63) the i^{th} column of the jacobian can be found to be

$$^0_nJ_i = \begin{bmatrix} z_i \times {}^0P^i_n \\ z_i \end{bmatrix} \tag{2.64}$$

Applying (2.62), (2.63), and (2.64) to the IRB1400 industrial robot, the equations presented in Figure 2.12 are obtained.

$$^0v_0 = 0 \quad ^0w_0 = 0 \quad ^0v_1 = 0 \quad ^0w_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} \quad ^0v_2 = \begin{bmatrix} -a_1s_1\dot{\theta}_1 \\ a_1c_1\dot{\theta}_1 \\ 0 \end{bmatrix} \quad ^0w_2 = \begin{bmatrix} s_1\dot{\theta}_2 \\ -c_1\dot{\theta}_2 \\ \dot{\theta}_1 \end{bmatrix}$$

$$^0v_3 = \begin{bmatrix} (a_2s_1s_2 - a_1s_1)\dot{\theta}_1 - a_2c_1c_2\dot{\theta}_2 \\ (a_1c_1 - a_2c_1s_2)\dot{\theta}_1 - a_2c_2s_1\dot{\theta}_2 \\ -a_2s_2\dot{\theta}_2 \end{bmatrix} \quad ^0w_3 = \begin{bmatrix} s_1(\dot{\theta}_2 + \dot{\theta}_3) \\ -c_1(\dot{\theta}_2 + \dot{\theta}_3) \\ \dot{\theta}_1 \end{bmatrix}$$

$$^0v_4 = \begin{bmatrix} (a_2s_2 - a_1 + a_3s_{23} - d_4c_{23})s_1\dot{\theta}_1 - (a_2c_2 + d_4s_{23} - a_3c_{23})c_1\dot{\theta}_2 - (d_4s_{23} + a_3c_{23})c_1\dot{\theta}_3 \\ (a_1 - a_2s_2 + d_4c_{23} - a_3s_{23})c_1\dot{\theta}_1 - (a_2c_2 + d_4s_{23} + a_3c_{23})s_1\dot{\theta}_2 - (d_4s_{23} + a_3c_{23})s_1\dot{\theta}_3 \\ (d_4c_{23} - a_3s_{23} - a_2s_2)\dot{\theta}_2 + (d_4c_{23} - a_3s_{23})\dot{\theta}_3 \end{bmatrix}$$

$$^0w_4 = \begin{bmatrix} s_1(\dot{\theta}_2 + \dot{\theta}_3) + c_1c_{23}\dot{\theta}_4 \\ -c_1(\dot{\theta}_2 + \dot{\theta}_3) + s_1c_{23}\dot{\theta}_4 \\ \dot{\theta}_1 + s_{23}\dot{\theta}_4 \end{bmatrix}$$

$$^0v_5 = \begin{bmatrix} (a_2s_2 - a_1 + a_3s_{23} - d_4c_{23})s_1\dot{\theta}_1 - (a_2c_2 + d_4s_{23} - a_3c_{23})c_1\dot{\theta}_2 - (d_4s_{23} + a_3c_{23})c_1\dot{\theta}_3 \\ (a_1 - a_2s_2 + d_4c_{23} - a_3s_{23})c_1\dot{\theta}_1 - (a_2c_2 + d_4s_{23} + a_3c_{23})s_1\dot{\theta}_2 - (d_4s_{23} + a_3c_{23})s_1\dot{\theta}_3 \\ (d_4c_{23} - a_3s_{23} - a_2s_2)\dot{\theta}_2 + (d_4c_{23} - a_3s_{23})\dot{\theta}_3 \end{bmatrix}$$

$$^0w_5 = \begin{bmatrix} s_1(\dot{\theta}_2 + \dot{\theta}_3) + c_1c_{23}\dot{\theta}_4 + (c_1s_{23}s_4 + s_1c_4)\dot{\theta}_5 \\ -c_1(\dot{\theta}_2 + \dot{\theta}_3) + s_1c_{23}\dot{\theta}_4 + (s_1s_{23}s_4 - c_1c_4)\dot{\theta}_5 \\ \dot{\theta}_1 + s_{23}\dot{\theta}_4 - c_{23}s_4\dot{\theta}_5 \end{bmatrix} \quad ^0v_6 = \begin{bmatrix} ^0v_6(x) \\ ^0v_6(y) \\ ^0v_6(z) \end{bmatrix}$$

$$^0v_6(x) = ((a_2s_2 - a_1 + a_3s_{23} - d_4c_{23})s_1 + d_6((s_1s_{23}c_4 + c_1s_4)s_5 - s_1c_{23}c_5))\,\dot{\theta}_1 +$$

$$+ ((-a_2c_2 - d_4s_{23} - a_3c_{23}) - d_6(c_{23}c_4s_5 + s_{23}c_5))c_1\,\dot{\theta}_2 + (c_1(-d_4s_{23} - a_3c_{23}) - d_6(c_{23}c_4s_5 +$$

$$+ s_{23}c_5))\,\dot{\theta}_3 + d_6(s_1c_4s_5 + c_1s_4s_5s_{23})\,\dot{\theta}_4 + d_6(s_1c_5s_4 - c_1c_{23}s_5 - c_1c_4c_5s_{23})\,\dot{\theta}_5$$

$$^0v_6(y) = ((a_1 - a_2s_2 + d_4c_{23} - a_3s_{23})c_1 + ((-c1*s23*c4+s1*s4)*s5+c1*c23*c5)*d_6)\dot{\theta}_1 - ((a_2c_2 + d_4s_{23} + a_3c_{23}) + d_6(c_{23}c_4s_5 + s_{23}c_5))s_1\dot{\theta}_2 - ((d_4s_{23} + a_3c_{23})s_1 + + d_6s_1(c_{23}c_4s_5 + s_{23}c_5))\dot{\theta}_3 + d_6(s_{23}s_1s_4s_5 - c_1c_4s_5)\dot{\theta}_4 - d_6(c_{23}s_1s_5 + c_1s_4c_5 + + s_1c_4c_5s_{23})\dot{\theta}_5$$

$$^0v_6(z) = ((c_{23}c_5 - s_{23}c_4s_5)d_6 + d_4c_{23} - a_3s_{23} - a_2s_2)\dot{\theta}_2 + ((c_{23}c_5 - s_{23}c_4s_5)d_6 + d_4c_{23} - a_3s_{23})\dot{\theta}_3 - s_5s_4c_{23}d_6\dot{\theta}_4 + (c_{23}c_5c_4 - s_5s_{23})d_6\dot{\theta}_5$$

$$^0w_6 = \begin{bmatrix} s_1(\dot{\theta}_2+\dot{\theta}_3)+c_1c_{23}\dot{\theta}_4+(c_1s_{23}s_4+s_1c_4)\dot{\theta}_5+((-c_1s_{23}c_4+s_1s_4)s_5+c_1c_{23}c_5)\dot{\theta}_6 \\ -c_1(\dot{\theta}_2+\dot{\theta}_3)+s_1c_{23}\dot{\theta}_4+(s_1s_{23}s_4-c_1c_4)\dot{\theta}_5-((s_1s_{23}c_4+c_1s_4)s_5-s_1c_{23}c_5)\dot{\theta}_6 \\ \dot{\theta}_1+s_{23}\dot{\theta}_4-c_{23}s_4\dot{\theta}_5+(c_{23}c_4s_5+s_{23}c_5)\dot{\theta}_6 \end{bmatrix}$$

$$^0_4J = \begin{bmatrix} (a_2s_2-a_1+a_3s_{23}-d_4c_{23})s_1 & -(a_2c_2+d_4s_{23}-a_3c_{23})c_1 & -(d_4s_{23}+a_3c_{23})c_1 & 0 \\ (a_1-a_2s_2+d_4c_{23}-a_3s_{23})c_1 & -(a_2c_2+d_4s_{23}+a_3c_{23})s_1 & -(d_4s_{23}+a_3c_{23})s_1 & 0 \\ 0 & d_4c_{23}-a_3s_{23}-a_2s_2 & d_4c_{23}-a_3s_{23} & 0 \\ 0 & s_1 & s_1 & c_1c_{23} \\ 0 & -c_1 & -c_1 & s_1c_{23} \\ 1 & 0 & 0 & s_{23} \end{bmatrix}$$

$$^0_3J = \begin{bmatrix} a_2s_1s_2-a_1s_1 & -a_2c_1c_2 & 0 \\ a_1c_1-a_2c_1s_2 & -a_2c_2s_1 & 0 \\ 0 & -a_2s_2 & 0 \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \quad ^0_6J = \begin{bmatrix} J_{11} & J_{12} & J_{13} & J_{14} & J_{15} & J_{16} \\ J_{21} & J_{22} & J_{23} & J_{24} & J_{25} & J_{26} \\ J_{31} & J_{32} & J_{33} & J_{34} & J_{35} & J_{36} \\ J_{41} & J_{42} & J_{43} & J_{44} & J_{45} & J_{46} \\ J_{51} & J_{52} & J_{53} & J_{54} & J_{55} & J_{56} \\ J_{61} & J_{62} & J_{63} & J_{64} & J_{65} & J_{66} \end{bmatrix}$$

$\mathbf{J_{11}}$ = (a2*s2 - a1 + a3*s23 - d4*c23)*s1 + d6*((s1*c4*s23 + c1*s4)*s5 –
- s1*c23*c5);
$\mathbf{J_{12}}$ = ((-a2*c2 - d4*s23 - a3*c23) - d6*(c23*c4*s5 + s23*c5))*c1;
$\mathbf{J_{13}}$ = c1*((-d4*s23 - a3*c23) - d6*(c23*c4*s5 + s23*c5));
$\mathbf{J_{14}}$ = d6*(s1*c4*s5 + c1*s4*s5*s23);
$\mathbf{J_{15}}$ = d6*(s1*c5*s4 - c1*c23*s5 - c1*c4*c5*s23);
$\mathbf{J_{16}}$ = 0;

$\mathbf{J_{21}}$ = (a1 - a2*s2 + d4*c23 - a3*s23)*c1 +
+((-c1*s23*c4+s1*s4)*s5+c1*c23*c5)*d6;
$\mathbf{J_{22}}$ = - ((a2*c2 + d4*s23 + a3*c23) + d6*(c23*c4*s5 + s23*c5))*s1;

```
J23 = - (d4*s23 + a3*c23)*s1 - d6*s1*(c23*c4*s5 + s23*c5);
J24 = d6*(s23*s1*s4*s5 - c1*c4*s5);
J25 = - d6*(c23*s1*s5 + c1*s4*c5 + s1*c4*c5*s23);
J26 = 0;

J31 = 0;
J32 = (c23*c5 - s23*c4*s5)*d6 + d4*c23 -a3*s23 -a2*s2;
J33 = (c23*c5 - s23*c4*s5)*d6 +d4*c23 -a3*s23;
J34 = - s5*s4*c23*d6;
J35 = (c23*c5*c4 - s5*s23)*d6;
J36 = 0;

J41 = 0;
J42 = s1;
J43 = s1;
J44 = c1*c23;
J45 = c1*s23*s4 + s1*c4;
J46= (- c1*s23*c4 + s1*s4)*s5 + c1*c23*c5;

J51 = 0;
J52 = - c1;
J53 = - c1;
J54 = s1*c23;
J55 = s1*s23*s4 - c1*c4;
J56 = - ((s1*s23*c4 + c1*s4)*s5 - s1*c23*c5);

J61 = 1;
J62 = 0;
J63 = 0;
J64 = s23;
J65 = -c23*s4;
J66 = c23*c4*s5 + s23*c5;
```

Note: These calculations were made in *MatLab* using the symbolic Toolbox.

Figure 2.12 Linear and angular velocities, jacobian matrices ${}^0_3J$, ${}^0_4J$ and ${}^0_6J$

2.4 Singularities

If the objective is to use the differential kinematics equation (2.28) for simplicity and efficiency, then it's necessary to deal with the singularities of the jacobian. The differential kinematics equation maps the vector of joint velocities $\dot{q} = [\dot{q}_1 \quad \dot{q}_2 \quad \dot{q}_3 \quad \dot{q}_4 \quad \dot{q}_5 \quad \dot{q}_6]^T$ with the *end-effector* twist vector $V = \begin{bmatrix} v^T & w^T \end{bmatrix}^T$. This mapping is seriously affected when the jacobian is rank-deficient (kinematics

singularities), because in those situations the mobility of the robot is reduced, the inverse kinematics may show infinite solutions, and (because the jacobian determinant may take very small values near singularities) small task space velocities may cause very large joint velocities [2]. So, to control the robot manipulator it is necessary to find all singular configurations and design a scheme to identify a singular configuration approach.

In order to find all the singular points of the ABB IRB 1400 anthropomorphic industrial robot, which has a very simple kinematic structure, a scheme will be used that separates the arm singularities and the wrist singularities. By dividing the jacobian into four 3×3 blocks it can then be expressed as

$$ {}^{0}_{6}J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{2.65} $$

Now, looking to all the elements of J_{12} (Figure 2.12) it is clear that $\det(J_{12})$ vanishes making $d_6=0$. That is equivalent to choosing the origin of the *end-effector* frame coincident with the origin of axis 4 and 5, i.e., making $\mathbf{p}_w = \mathbf{p}$. Since singularities are a characteristic of the robot structure and do not depend on the frames chosen to describe kinematically the robot, this procedure is allowed. It's possible then to write

$$ \det(J) = \det(J_{11}) * \det(J_{22}) \tag{2.66} $$

The robot's singular configurations are the ones that make $\det(J) = 0$ which means from (2.66)

$$ \det(J_{11}) = 0 \quad \text{or} \quad \det(J_{22}) = 0 \tag{2.67} $$

Solving the first equation leads to the so called *arm singularities* and solving the second leads to the *wrist singularities.*

Wrist Singularities

The wrist singularities can be found just by analyzing the structure of $\det(J_{22})$:

$$ \det(J_{22}) = \det\begin{pmatrix} z_4 & z_5 & z_6 \end{pmatrix} = \det\begin{pmatrix} c_1c_{23} & c_1s_{23}s_4 - c_1c_4 & (s_1s_4 - c_1s_{23}c_5)s_5 + c_1c_{23}c_5 \\ s_1c_{23} & s_1s_{23}s_4 - c_1c_4 & -(s_1s_{23}c_4 + c_1s_4)s_5 + s_1c_{23}c_5 \\ s_{23} & -c_{23}s_4 & c_{23}c_4s_5 + s_{23}c_5 \end{pmatrix} \tag{2.68} $$

The above determinant is non-null if the column vectors of J_{22} (which correspond to z_4, z_5, and z_6) are linearly independent, i.e., the singular configurations are the ones that make at least two of them linearly dependent. Now, vectors z_4 and z_5 are linearly independent in all configurations, and the same occurs between z_5 and z_6. This conclusion is easy to understand looking to (2.68) and/or remembering that z_4

is perpendicular to z_5, and z_5 is perpendicular to z_6 in all possible robot configurations. A singular configuration appears when z_4 and z_6 are linearly dependent, i.e., when those axis align with each other, which means s_5=0 from (2.68). Consequently the wrist singular configurations occur when,

$$\theta_5 = 0 \quad \text{or} \quad \theta_5 = \pi \tag{2.69}$$

The second condition ($\theta_5 = \pi$) is out of joint 5 work range, and because of that is of no interest, i.e., the wrist singularities will occur whenever $\theta_5 = 0$.

Arm Singularities

The arm singularities occur when $\det(J_{11}) = 0$ making again $\mathbf{p} = \mathbf{p}_w \Rightarrow d_6 = 0$, i.e., when

$$\det\begin{pmatrix} (a_2s_2 - a_1 + a_3s_{23} - d_4c_{23})s_1 & -(a_2c_2 + d_4s_{23} - a_3c_{23})c_1 & -(d_4s_{23} + a_3c_{23})c_1 \\ (a_1 - a_2s_2 + d_4c_{23} - a_3s_{23})c_1 & -(a_2c_2 + d_4s_{23} + a_3c_{23})c_1 & -(d_4s_{23} + a_3c_{23})c_1 \\ 0 & d_4c_{23} - a_3s_{23} - a_2s_2 & d_4c_{23} - a_3s_{23} \end{pmatrix} = 0 \tag{2.70}$$

Solving (2.70) gives

$$-a_2(d_4c_3 - a_3s_3)(a_3s_{23} - d_4c_{23} + a_2s_2 - a_1) = 0 \tag{2.71}$$

which leads to the following conditions:

$$-a_3s_3 + d_4c_3 = 0$$

and/or

$$a_3s_{23} - d_4c_{23} + a_2s_2 - a_1 = 0 \tag{2.72}$$

The first condition leads to $\theta_3 = \text{arctg}\left(\frac{d_4}{a_3}\right)$. The elbow is completely stretched out and the robot manipulator is in the so called *elbow singularity*. This value of θ_3 is out of joint 3's work range, so it corresponds to a non-reachable configuration, and because of that is of no interest.

The second condition corresponds to configurations in which the origin of the wrist (origin of axis 4) lies in the axis of joint 1, i.e., lies in z_1 (note that z_1 is coincident with z_0). In those configurations, the position of the wrist cannot be changed by rotation of the remaining free joint θ_1 (remember that an anthropomorphic manipulator with a spherical wrist uses the anthropomorphic arm to position the spherical wrist, which is then used to set the orientation of the *end-effector*). The manipulator is in the so called *shoulder singularity*.

In conclusion, the arm singularities of the ABB IRB 1400 industrial robot are confined to all the configurations that correspond to a shoulder singularity, i.e., to configurations where $a_3 s_{23} - d_4 c_{23} + a_2 s_2 - a_1 = 0$.

2.4.1 Brief Overview: Singularity Approach

As already mentioned, the solutions of the inverse kinematics problem can be computed from

$$\dot{q} = J^{-1}(\theta)V \tag{2.73}$$

solving (2.28) in order to $\dot{q}$. With this approach it's possible to compute the joint trajectories (q, $\dot{q}$), initially defined in terms of the *end-effector* wrist vector V and of the initial position/orientation. In fact, if q(0) is known it's possible to calculate:

$\dot{q}$ (t) from: $\dot{q}(t) = J^{-1}(\theta)V(t)$

and

q(t) from: $$q(t) = q(0) + \int_0^t \dot{q}(\alpha)d\alpha \tag{2.74}$$

Nevertheless, this is only possible if the jacobian is full rank, i.e., if the robot manipulator is out of singular configurations where the jacobian contains linearly dependent column vectors. In the neighborhood of a singularity, the jacobian inverse may take very high values, due to small values of det(J), i.e., in the neighborhood of a singular point small values of the velocity in the task space (V) can lead to very high values of the velocity in the joint space ($\dot{q}$).

The *singular value decomposition* (SVD) of the jacobian [3,8-10] is maybe the most general way to analyze what happens in the neighborhood of a singular point; also it is the only general reliable method to numerically determine the rank of the jacobian and the closeness to a singular point. With the inside given by the SVD of the jacobian, a *Damped Least-Square* scheme [9] can be optimized to be used in near-singular configurations. The *Damped Least-Square* (DLS) scheme trades-off accuracy of the inverse kinematics solutions with feasibility of those solutions: this trade-off is regulated by the damping factor ξ. To see how this works, let's define the DLS inverse jacobian by rewriting (2.28) in the form

$$(JJ^T + \xi^2 I)\dot{q} = J^T V \tag{2.75}$$

where ξ is the so-called damping factor. Solving (2.75) in order to $\dot{q}$ gives

$$\dot{q} = (JJ^T + \xi^2 I)^{-1} J^T V = J_{dls}^{-1} V \tag{2.76}$$

with J_{dls} being the *damped least-square* jacobian inverse. The solutions of (2.76) are the ones that minimize the following cost function [2,9,11]:

$$g(\dot{q}) = \frac{1}{2}(V - J\dot{q})^T (V - J\dot{q}) + \frac{1}{2}\xi^2 \dot{q}^T \dot{q} \tag{2.77}$$

resulting from the condition

$$\min_{\dot{q}} \left(\|V - J\dot{q}\|^2 + \xi^2 \|\dot{q}\|^2 \right) \tag{2.78}$$

The solutions are a trade-off between the *least-square* condition and the minimum norm condition. It is very important to select carefully the damping factor ξ : small values of ξ lead to accurate solutions but with low robustness to the singular or near-singular occurrences (= high degree of failure in singular or near-singular configurations), i.e., low robustness to the main reason to use the scheme. High values of ξ lead to feasible but awkward solutions.

To understand how to select the damping factor ξ, in the following the jacobian will be decomposed using the SVD technique. The SVD of the jacobian can be expressed as

$$J = U\Sigma V^T = \sum_{1}^{6} \sigma_i u_i v_i^T \tag{2.79}$$

where $\sigma_1 > \sigma_2 > \ldots > \sigma_r > 0$ (r = rank(J)) are the jacobian *singular values* (positive square roots of the eigenvalues of J^TJ), v_i (columns of the orthogonal matrix V) are the so-called *right or input singular vectors* (orthonormal eigenvectors of J^TJ) and u_i (columns of the orthogonal matrix U) are the so-called *left or output singular vectors* (orthonormal eigenvectors of JJ^T). The following properties hold:

$$R(J) = \text{span } \{u_1, \ldots, u_r\}^6$$
$$N(J) = \text{span } \{v_{r+1}, \ldots, v_6\}$$

The range of the jacobian R(J) is the set of all possible task velocities, those that could result from all possible joint velocities: $R(J) = \{V \in \Re^6 : V = J\dot{q} \text{ for all possible } \dot{q} \in \Re^6\}$. The first r input singular vectors constitute a base of R(J). So, if in a singularity the rank of the jacobian is reduced then one other effect of a singularity will be the decrease of dim[R(J)] by eliminating a linear combination of

---

[6] The span of $\{a_1, \ldots a_n\}$ is the set of the linear combinations of $a_1, \ldots a_n$.

task velocities from the space of feasible velocities, i.e., the reduction of the set of all possible task velocities.

The null space of the jacobian N(J) is the set of all the joint velocities that produce a null task velocity at the current configuration: $N(J) = \{ \dot{q} \in \Re^6 : J\dot{q} = 0 \}$. The last (6-r) output singular vectors constitute a base of N(J). So, in a singular configuration the dimension of N(J) is increased by adding a linear combination of joint velocities that produce a null task velocity.

Using the SVD of the jacobian (2.78) in the DLS form of the inverse kinematics (2.75) results in

$$\dot{q} = \sum_{1}^{6} \frac{\sigma_i}{\sigma_i^2 + \xi^2} v_i u_i^T V \tag{2.80}$$

The following properties hold:

$$\begin{aligned} R(J_{lds}) &= R(J^\dagger)^7 = N^\perp(J)^8 = \text{span } \{u_1, \ldots, u_r\} \\ N(J_{lds}) &= R(J^\dagger) = R^\perp(J)^9 = \text{span } \{v_{r+1}, \ldots, v_6\} \end{aligned} \tag{2.81}$$

which means that the properties of the *damped least-squares* inverse solution are analogous to those of the pseudoinverse solution (remember that the inverse pseudoinverse solution gives a *least-square* solution with a minimum norm to equation (2.28)).

The damping factor has little influence on the components for which $\sigma_i >> \xi$ because in those situations

$$\frac{\sigma_i}{\sigma_i^2 + \xi^2} \approx \frac{1}{\sigma_i} \tag{2.82}$$

i.e., the solutions are similar to the pure *least-square* solutions.

Nevertheless, when a singularity is approached, the smallest singular value (the r-th singular value) tend's to zero, the associated component of the solution is driven to zero by the factor $\frac{\sigma_i}{\xi^2}$ and the joint velocity associated with the near-degenerate components of the commanded velocity V are progressively reduced, i.e., at a singular configuration, the joint velocity along v_r is removed (no longer remains in the null-space of the jacobian as in the pure *Least-Square* solution) and the task

---

[7] $J^\dagger$ is the pseudoinverse jacobian.
[8] Orthogonal complement of the null space joint velocities.
[9] Orthogonal complement of the feasible space task velocities.

velocity along u_r becomes feasible. That is how the damping factor works; as a measure or indication of the degree of approximation between the damped and pure *least-square* solutions. Then a strategy [8], initially presented by [12], can be used to adjust ξ as a function of the closeness to the singularity. Based on the estimation of the smallest singular value of the jacobian, we can define a singular region and use the exact solution (ξ=0) outside the region and a damped solution inside the region. In this case, a varying ξ should be used (increasing as we approach the singular point) to achieve better performance (as mentioned the damped solutions are different from the exact solutions). The damping factor ξ can then be defined using the following law modified from [9]

$$\xi^2 = \begin{cases} 0 & \hat{\sigma}_6 \geq \varepsilon \\ \left(1 - \left(\frac{\hat{\sigma}_6}{\varepsilon}\right)^{2\eta}\right)\xi^2_{max} & \hat{\sigma}_6 < \varepsilon \end{cases} \tag{2.83}$$

where ξ^2_{max} and η are defined by the user to shape the solution to his needs, ε defines the size of the region and $\hat{\sigma}_6$ is the estimate of the smallest singular value. The estimate is done using a recursive algorithm originally presented at [13] and later extended by [14] to estimate not only the smallest singular value but also the second smallest singular value. This procedure avoids estimation inaccuracy due to the cross of the two smallest singular values, when the manipulator approaches both the wrist and the shoulder singularity. The algorithm is as follows:

Suppose we have estimates of the two last input singular vectors $\hat{v}_5$ and $\hat{v}_6$ with

$$\begin{aligned} \hat{v}_5 &\approx v_5 \text{ and } \|\hat{v}_5\| = 1 \\ \hat{v}_6 &\approx v_6 \text{ and } \|\hat{v}_6\| = 1 \end{aligned} \tag{2.84}$$

The estimate $\hat{v}_6$ is then use to compute $\hat{v}'_6$ from

$$\left(J^T J + \xi^2 I\right)\hat{v}'_6 = \hat{v}_6 \tag{2.85}$$

Then the estimate $\hat{\sigma}^2_6$ is computed from

$$\hat{\sigma}^2_6 = \frac{1}{\|\hat{v}'_6\|} - \xi^2 \tag{2.86}$$

and the initial estimate $\hat{v}_6$ is updated using

$$\hat{v}_6 = \frac{\hat{v}_6'}{\left\|\hat{v}_6'\right\|} \tag{2.87}$$

The second smallest singular value is computed using the estimate $\hat{v}_6$ from,

$$\left[J^T J + \xi^2 I - \left(\hat{\sigma}_6^2 + \xi^2\right)\hat{v}_6\hat{v}_6^T\right]\hat{v}_5' = \hat{v}_5 \tag{2.88}$$

Then the estimate $\hat{\sigma}_5^2$ is computed from

$$\hat{\sigma}_5^2 = \frac{1}{\left\|\hat{v}_5'\right\|} - \xi^2 \tag{2.89}$$

and finally the initial estimate $\hat{v}_5$ is updated using

$$\hat{v}_5 = \frac{\hat{v}_5'}{\left\|\hat{v}_5'\right\|} \tag{2.90}$$

Special care should be taken with the numerical implementation of the DLS inverse kinematics solutions, to correct the numerical drift. Basically a feedback term can be used [2,9,15] by making

$$V = V_d + K.e = V_d + K\begin{pmatrix} \mathbf{p}_d - \mathbf{p} \\ \frac{1}{2}(\mathbf{n}\times\mathbf{n}_d + \mathbf{s}\times\mathbf{s}_d + \mathbf{a}\times\mathbf{a}_d) \end{pmatrix} \tag{2.91}$$

where K is a positive definite diagonal 6×6 matrix, $\mathbf{p}_d$ and $\mathbf{p}$ are the desired and actual position, and the orientation is defined in terms of the desired and actual ($\mathbf{n}$, $\mathbf{s}$, $\mathbf{a}$) vectors of the *end-effector* frame.

Due to the increase of *end-effector* errors [11] in the neighborhood of a singularity by means of the near-degenerate components of *end-effector* velocity, the matrix K should be corrected using $K=\rho K_0$, where K_0 is a diagonal constant matrix and ρ is the correcting factor. Now, inside a singular region we should use $K=0$ because in some situations the resulting joint velocities can drive the manipulator to reach the joint limits, even if eventually the error will approach zero. When the manipulator is sufficiently away from a singularity, we should have $\rho=1$. So, generally we define ρ as

$$\rho = \begin{cases} 0 & \sigma_6 \le \varepsilon \\ \dfrac{(\sigma_6 - \varepsilon)^2}{(n-1)^2 \varepsilon^2} & \varepsilon < \sigma_6 < n\varepsilon \\ 1 & \text{otherwise} \end{cases} \tag{2.92}$$

where **n** is defined by the user based on self-experience and on test results with a particular robotic manipulator setup.

2.5 Position Sensing

The IRB1400 uses resolvers [16-19] as position sensors. The drive unit used at this robot (manufactured by *ELMO AB* for *ABB Robotics*), includes a PM AC synchronous motor, both current feedback devices, a brake, and a brushless resolver, all assembled at factory ,i.e., they come in one piece [20].

A brushless resolver consists of a stator, a rotor and a rotary transformer. The stator and rotor windings are distributed in a way that the magnetic flux is distributed as a sine wave of the angle of rotation (perfect resolver). The output of a resolver is therefore an AC voltage in accordance with the angular position of the shaft. This type of position sensor is characterized by its high accuracy output, maintenance free brushless design, and immunity to noise, vibration, and shock. Other characteristics introduced by highly automated manufacturer production facilities include homogeneity in accuracy, transformation ratio, phase-shift, etc.

These characteristics significantly reduce major sources of error such as:

1. Amplitude imbalance due to different amplitudes of the resolver output signals
2. Imperfect quadrature due to phase-shift
3. Inductance harmonic error due to imperfect inductance profiles, i.e., the inductance profiles do not follow perfect sine wave as consequence of imperfect sinusoidal winding

Two types of resolvers can be considered (Figure 2.13): *Brushless Amplitude Output Resolvers* (BAOR) and *Brushless Phase-Shift Output Resolvers* (BPOR)[10]. Resolvers of type BAOR are excited by an AC voltage to the rotor winding and the output is obtained from the stator windings in the form sine and cosine voltages proportional to the rotation angle θ. Resolvers of the type BPOR are excited by sine and cosine voltages to the stator windings and the output is obtained from the

[10] *Tamagawa Seiki Co. LTD.* names these resolvers as BRX and BRT, respectively.

rotor winding in the form of a sine voltage with phase-shifted in proportion to the rotation angle θ.

The IRB 1400 uses BAOR type resolvers from the Japanese manufacturer Tamagawa Seiki Co. LTD. [19,20].

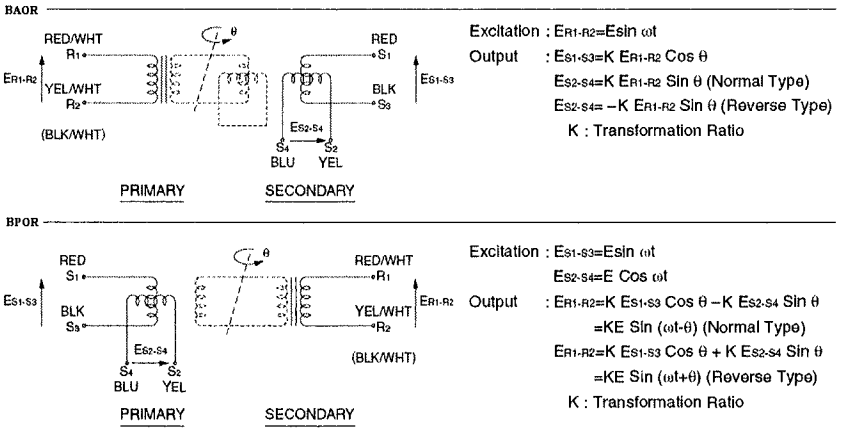


Figure 2.13 Types of resolvers

The use of a resolver implies the availability of a *resolver to digital converter* (RDC) and processing circuit [21-23]. The RDC is used to track and convert resolver signals to a digital parallel binary word, generally using a ratiometric tracking conversion method that improves noise immunity and tolerance to lead length (important when the converter is remote from the resolver). The RDC circuit uses an RDC along with the necessary interface and signal conditioning circuitry. Because noise can degrade significantly the accuracy of the measurement, special care must be taken with the driving lines from and to the resolvers: the use of shielded twisted pair cabling and isolation amplifiers may be needed.

The basic functional diagram of an RDC is presented in Figure 2.14, where it is used data relative to the *Analog Devices* RDC model AD2S80A. The converter works as a type II closed-loop system with the angle ϕ as a control variable (this angle is the current converter estimate of the angle θ).

Generally, the converter's functioning can be described as follows: First the inputs (resolver outputs $E_{S2\text{-}S4}$ and $E_{S1\text{-}S3}$) are multiplied by cos(ϕ) and sin(ϕ), respectively, at the *ratio multiplier*. Then the difference between the signals is computed giving the ratio multiplier output AC error signal $E_{ac} = A_1.K.E.\sin(\theta - \phi).\sin(\omega t)$, where A_1 is the ratio multiplier gain (fixed at 14.5 for AD2S80A). Second, this error signal is synchronously demodulated at the *phase sensitive demodulator* (PSD), using the resolver excitation frequency as a demodulation reference, leaving the error signal

$E_{PSD} = A_1.K.E\sin(\theta - \phi)$. The output of the demodulator is a DC voltage proportional to the RMS value of the demodulator input: $\frac{\pm\sqrt{2}}{\pi} * Demodulator\_Input_{RMS}$ (for sinusoidal input signals in phase or antiphase with the reference signal). Before entering the PSD, the signal passes over an *HF filter* (with components selected by the user) to remove any DC offset voltage. Then the PSD output passes through the *integrator* (with components selected by the user), whose output signal (proportional to the velocity of the resolver) is fed to the *voltage controlled oscillator* (VCO). The VCO integrates the velocity signal and compares the resulting signal with the minimum DC voltage resolution (uses two comparators for positive and negative voltages, meaning rotation in the positive direction or in the negative direction) and updates the up/down counter by producing the counter clock and direction signal. The value of the internal latch used to interface with the user is also updated with the counter value. An RDC works similarly to a *successive approximation* type *analog to digital converter*.

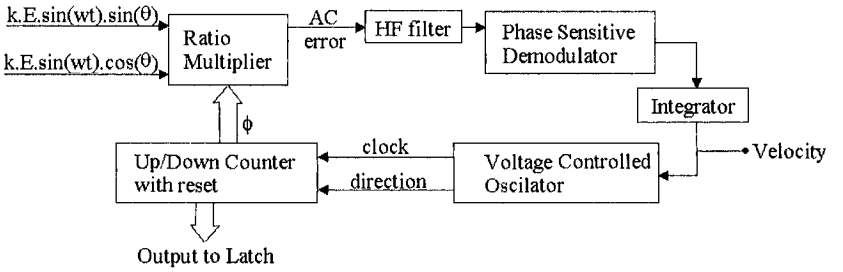


Figure 2.14 Resolver to digital converter basic functional diagram

The RDC returned digital value is generally a 12, 14, or 16-bit binary number containing the actual rotation angle. This angle should be mapped to the robot's join space. For that, the following guidelines should be used:

1. Choose an angle data format, i.e., degrees, radians
2. Account for the resolver offset[11], i.e., the resolver reading when the manipulator is in the home position. At that point, we should have *number_of_rotations* = 0 and *actual_angle* = 0

A complete RDC circuit implementation should also save the total number of rotations in an 8-bit *up-down* counter/register. In essence, the circuit should give the rotation angle of the motor in the actual rotation and the total number of rotations already performed.

[11] Usually these values are measured by the robot manufacturer and printed on the robot or in the robot documentation.

2.6 Actuators: Motors

Generally the actuators used to move the joints of any industrial robot are motors, usually DC permanent magnet (PM) motors or AC PM motors. Other motors can be used, including pneumatic or hydraulic servo motors. The IRB 1400 uses three-phase synchronous AC PM motors, with six poles (axes 1-3) and four poles (axes 4-6), manufactured by *Elmo AB* – Sweden.

The three-phase synchronous AC PM motor rotating magnetic field is obtained by making a three-phase current to flow in the stator coil (Figure 2.15), which has a sinusoidal distribution. So, a brushless sine wave PM AC synchronous motor is obviously not mechanically commutated (there are no brushes) but instead the commutation is done by acting on the three-phase current signals. Nevertheless, the commutation position of the motor should be retained, i.e., the resolver reading when the motor is at the electrical home position (electrical 0° position) - this value is called the commutation offset (COMMOFF).

The usual procedure to find the commutation offsets is as follows:

1. Turn the motor to the commutating position by feeding a positive constant current to the motor
2. Feed the resolver with the necessary excitation signal (4kHz and 5 V_{rms} for IRB 1400 drives)
3. Adjust the resolver to +90° (± 0,5°), i.e, turn to the maximum value on coil Y of the resolver with the same phase as the 5V feeding signal. At that point we should have:
   Voltage across coil X = 0V
   and
   Voltage across coil Y = input voltage * transformation ratio

The value of the rotation angle (90 degrees) is the commutation offset. This procedure is used with the IRB 1400 drives, so that is why the COMMOFFS are constant for all drives (1.570800 radians). For some older robots, like the ABB IRB 2000 (up to model M90), the motor and the resolver are separate parts, assembled together by the manufacturer without following the above referred procedure. So, the COMMOFFS are different for all drives. The values are obtained at factory and printed on the robot or in the documentation; nevertheless, these values can be updated using the robot controller.

A full description of a three-phase synchronous sine wave PM motor can be found in:

1. *Design of Brushless Permanent-Magnet Motors*, Herdershot Jr., Magna Physics Publishing and Clarendon Press, Oxford, 1995, Chapters 6 and 7
2. *Electric Drives and their Controls*, R.M. Crowder, Clarendon Press, Oxford, 1995, Chapter 5, section 5.3

Nevertheless, a brief overview is presented here.

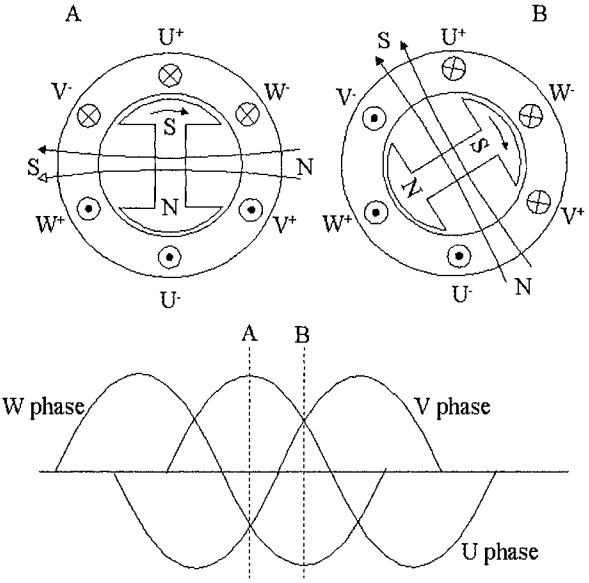


Figure 2.15 Three-phase synchronous motors and current signals

Considering β as the angle between rotor magnet north axis and the stator windings axis, it can be shown [17] that the motor torque is

$$T \propto \sin(\beta) \tag{2.93}$$

Consequently, the angle β must be kept at 90° in order to maximize the torque, which is done by phasing the current waveforms relative to the actual rotor position. To ensure that the ampere-conductor distribution remains in synchronism with the rotor's magnetic field, the stator supply frequency (f) must be equal to the rotor angular velocity (w_s), $w_s = 2.\pi.f$, which is related to the mechanical angular velocity of the motor (w_m) by $w_m = w_s/p$, where p is the number of the motor pole pairs. In order to keep the torque angle constant, i.e., to keep the ampere-distribution north axis in synchronism with the rotor north axis (displaced by 90°), a high-performance and precise sensor should be used (generally a resolver).

With this type of control action the motor follows the equation

$$\text{Torque} = \text{Flux} * \text{Current} \tag{2.94}$$

For this type of motors, the flux is constant, sinusoidally distributed in space, and the generated EMF varies sinusoidally in each phase. The overall torque-speed characteristic is presented in Figure 2.16. The maximum torque can be maintained

up to the base speed. After that, it is still possible to increase the velocity by changing β but the motor enters the field-weakening mode and any increase in speed is done at the expense of the peak torque.

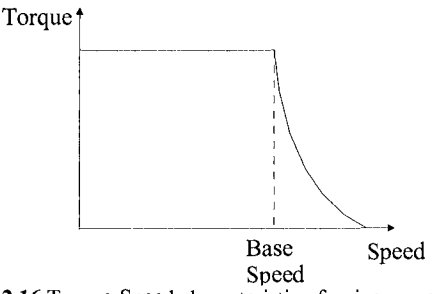


Figure 2.16 Torque-Speed characteristic of a sine wave motor

The "natural" relations for the back-EMF (E) and for the torque (T), used for a DC square wave motor still hold for a sine wave motor, i.e.,

$$\begin{aligned} T &= k_t * I \\ E &= k_e * w_m \end{aligned} \tag{2.95}$$

but now with $\frac{k_t}{k_e} = \frac{\sqrt{3}}{2} \neq 1$.

The torque constant (k_t) and the back-EMF constant (k_e) can be measured using the following equations:

$$k_e = \frac{\hat{e}_{LL}}{w_m} \text{ (V-s/rad)} \tag{2.96}$$

where $\hat{e}_{LL}$ is the peak line-line voltage and w_m is the mechanical angular velocity.

$$k_t = \frac{T}{\hat{i}} \text{ (Nm)} \tag{2.97}$$

where $\hat{i}$ is the peak line current when the motor is in normal operation, measured using a current sensor connected to measure the phase current directly and then displayed in an oscilloscope.

It is also possible to write

$$T.w_m = k_t * \hat{i} * \frac{\hat{e}_{LL}}{k_e} = \frac{\sqrt{3}}{2} * \hat{e}_{LL} * \hat{i} = \frac{\sqrt{3}}{2} * \sqrt{2} * E_{RMS} * \sqrt{2} * I_{RMS}$$
$$= \sqrt{3} * E_{RMS} * I_{RMS} = \text{Electrical} - \text{Mechanical Power Conversion} \quad (2.98)$$

and,

$$T = \frac{\sqrt{3} * E_{RMS} * I_{RMS}}{w_m} = k_t * I_{RMS} \Rightarrow k_t = \frac{\sqrt{3} * E_{RMS}}{2\pi * \frac{Vol_{RPM}}{60}} \quad (2.99)$$

2.6.1 Motor Drive System

In this section, the main circuits necessary to drive a three-phase AC synchronous PM motor are briefly presented. As already mentioned, a brushless AC PM motor requires alternating sine wave phase currents, because the motor is designed to generate sinusoidal back-EMF. The power electronic control circuit is very simple and uses some control strategy[12] to achieve torque, smooth speed, and accurate control, keeping the current to a safe value. In order to obtain sine wave phase currents, the power supply (DC voltage) must be switched on and off at high frequency, under the control of a current regulator that forces the power transistors to switch on and off in a way that the average current is a sine wave. Basically, the sine wave reference signals could just be applied directly to the power transistors, after appropriate power amplification. However, that means using the power transistors in the proportional or linear region, which will increase the operating temperature due to the high power loss. The power loss is reduced by switching the transistors on and off by comparing the sine wave reference with a high frequency triangular carrier wave (PWM - *pulse width modulation* circuit). The frequency and amplitude of the triangular wave are kept constant. The comparator switches on the transistors when the values of the reference sine wave exceed those of the triangular wave; and switches them off when the inverse situation occurs (Figure 2.17). The duty ratio is then increased and decreased by the sine wave, centered by 50%. This procedure leads to a average sine wave output, because the output of the inverter feeding the power transistors is 0V when the duty ratio is 50%.

Special care should be taken in selecting the carrier frequency, because the power loss increases with increasing frequency and the motor speed response decreases with decreasing frequency. Torque and current ripples appear more frequently at higher frequencies as well.

---

[12] A set of rules that determine when the power transistors are switched on and off

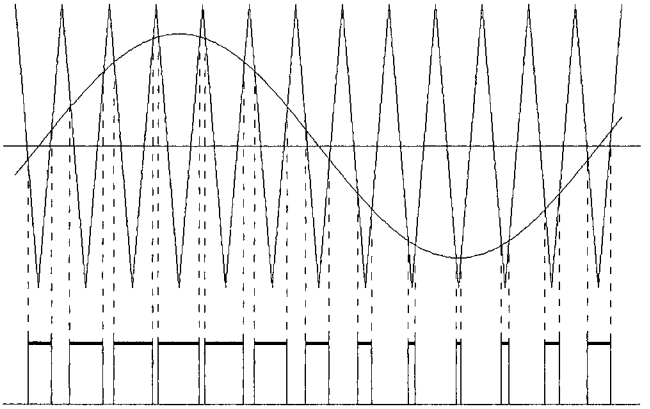
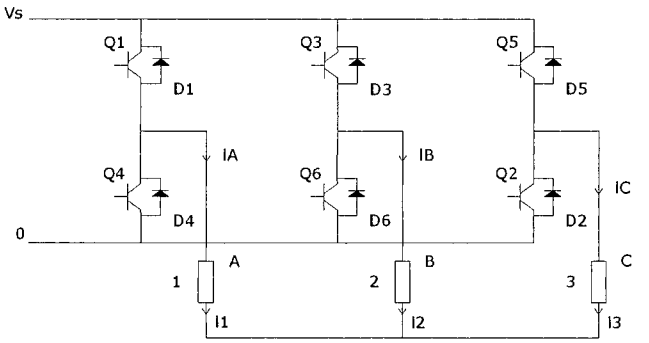


Figure 2.17 PWM basic functioning

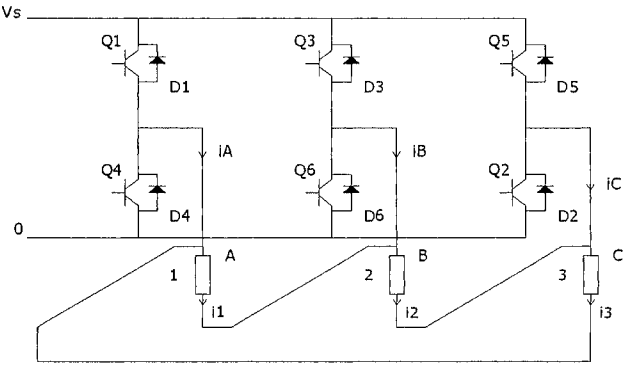
The basic power electronic circuit to control a sine wave three-phase AC PM motor is the full-bridge circuit. The transistors used in the circuit must have very low turn-on and turn-off switching times (of the order of nanoseconds) and some other properties summarized as follows:

1. Zero on-state forward voltage drop, to minimize losses and maximize available "voltage" to force current into the motor
2. Zero leakage current in the off state, to minimize losses because a power transistor usually has high voltages across it when it is off, so even a small leakage current can produce high losses in the transistor's off state
3. High forward-blocking capability that should be higher than the supply voltage by a safety margin (usually 30%). The reverse-blocking capability is generally a margin of the forward-blocking, usually because the power transistors are reverse-protected by appropriately connected diodes
4. High dv/dt capability, because modern power transistors are MOS-gated, with capacitive input impedance at the gate, which make's them sensitive to spurious turn-on when the gate is subjected to a high dv/dt. High dv/dt immunity is then desirable, but nevertheless a safe procedure is to drive the gate from a low impedance source/sink
5. High di/dt capability, to prevent current-crowding effects and second breakdown the di/dt capability must be high
6. High-speed switching, from transistors to minimize switching losses and also from the power diodes, because the commutation of inductive current from a transistor branch to a diode branch is the most important way to protect against destructive transient voltages

The full bridge circuit is presented in Figure 2.18 for two popular phase windings: eye and delta [17]. Figure 2.19 shows line current waveforms for three-phase sine wave motors, including transistor states and current paths.



Eye connected



Delta connected

Figure 2.18 Full bridge circuit for eye and delta connected windings

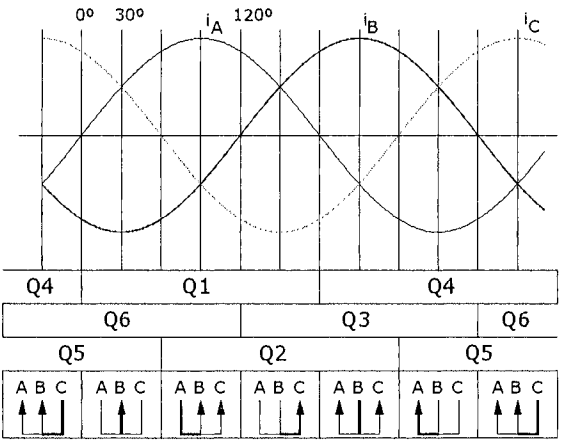


Figure 2.19 Line current waveforms for a sine wave motor, including transistor states and current paths

A general control system for a sine wave three-phase brushless motor is presented in Figure 2.20: includes a PWM circuit, over current (due to motor stall or short circuits) protection, a filter to damp DAC steps, a current controller (usually a PI controller designed to drive the motor current to the desired value) and a sine wave generator. Synchronization is achieved by changing current references in accordance with motor position.

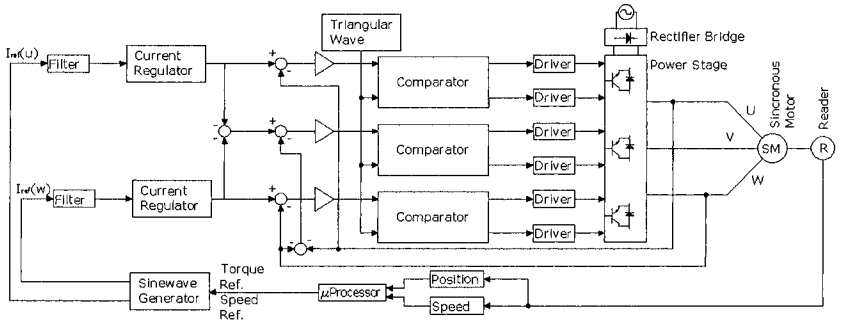


Figure 2.20 Block diagram of a general control system for a brushless synchronous three-phase sine wave motor

2.7 Dynamics

Dynamics deals with mapping forces exerted on the robot's parts as well as with the motion of the robot, i.e., its joint positions, velocities, and accelerations. This mapping is achieved using a set of mathematical equations, based on some specified dynamic formulation that describes the dynamic behavior of the robot manipulator, i.e., its motion. Those sets of equations constitute the dynamic model of the robot manipulator. The dynamic model can be used to simulate and control the robot manipulator, i.e., the dynamic model provides the means to compute the joint positions, velocities, and accelerations starting from the joint torques (*direct dynamics*), and the means to compute the joint torques using the joint positions, velocities, and accelerations (*inverse dynamics*).

The dynamic model is obtained starting from well known physical laws like the *Newtonian* mechanics and the *Lagrange* mechanics [6,24]. Several different dynamic formulations for robot manipulators were developed: *Lagrange-Euler*, *Newton-Euler*, *D'Alembert*, ... [1-3,7]. Nevertheless, they are equivalent to each other because they define the same physical phenomenon, i.e., the dynamics of rigid bodies assembled together to constitute a robot. Obviously, the structure of the motion equations is much different because each formulation was developed to achieve different objectives such as computation efficiency, simplicity to analyze and/or to simulate the structure, etc.

In this section, the dynamic model of the ABB IRB 1400 industrial robot will be briefly summarized using the Newton-Euler dynamic formulation. In the process, the other dynamic formulations are presented and briefly discussed.

2.7.1 Inertia Tensor and Mass Distribution

The mass distribution of a rigid body may be characterized by its inertial mass, for the case of one degree of freedom motions, and by its first moment of inertia, for simple rotations, i.e., rotations about a single axis. If there is more than one axis of rotation, the above properties are no longer suitable to characterize the mass distribution of the moving rigid body [6,24]. This is the case of a rigid robot manipulator, which is made by a series of rigid bodies, whose motion is 3-dimensional and therefore an infinite number of rotation axes is possible. The concept of *inertia tensor* is used in this case, which can be considered as a generalization of the concept of moment of inertia. If $\rho(x,y,z)$ is the mass density of a rigid body, then the inertia tensor may be defined as

$$I = \iiint \rho(r^2\mathbf{1} - rr)dv \tag{2.100}$$

where $\mathbf{1}$ is a unity tensor. The inertia tensor is a 3×3 matrix expressed in terms of some frame {A}

$${}^{A}I = \begin{bmatrix} I_{xx} & I_{yx} & I_{zx} \\ I_{xy} & I_{yy} & I_{zy} \\ I_{xz} & I_{yz} & I_{zz} \end{bmatrix} \tag{2.101}$$

where the diagonal elements are the moments of inertia about the axes x, y and z of frame {A}

$$\begin{aligned} I_{xx} &= \iiint \rho(y^2 + z^2)dv \\ I_{yy} &= \iiint \rho(z^2 + x^2)dv \\ I_{zz} &= \iiint \rho(x^2 + y^2)dv \end{aligned} \tag{2.102}$$

and the other elements (non-diagonal) are the products of inertia

$$\begin{aligned} I_{xy} &= I_{yx} = -\iint \rho xydv \\ I_{yz} &= I_{zy} = -\iint \rho yzdv \\ I_{zx} &= I_{xz} = -\iint \rho zxdv \end{aligned} \tag{2.103}$$

2.7.1.1 Important Results [6]
Next some important results will be presented, considering that the frame associated to the rigid body is {B} and the inertial frame is {A}.

Suppose that I is the inertia tensor of the rigid body expressed in terms of some reference frame. The moment of inertia about any axis of rotation **n** (different from any of the rigid body symmetry axes) with the same origin of the reference frame is

$$I_n = n^T I.n \tag{2.104}$$

Extension of the Parallel Axis Theorem This theorem is used here to compute the inertia tensor variation with linear motions of the reference frame. Suppose that {C} is the frame associated with the rigid body center of mass, {G} is some frame obtained from {C} by linear motion, and CP is the position vector of the center of mass expressed in terms of {G}. Then

$$I_G = I_C + M\,({}^{C}P^{T}\,{}^{C}P\,I_3 - {}^{C}P\,{}^{C}P^{T}) \tag{2.105}$$

where ${}^{C}P = (x_c, y_c, z_c)^T$ and I_3 is a 3×3 identity matrix.

If the rigid body is rotating, the inertia tensor expressed in terms of {A} ${}^{A}I$ is also varying with time, but the inertia tensor expressed in terms o {B} ${}^{B}I$ remains constant (remember that {B} is the frame associated with the rigid body). If the inertia tensor ${}^{B}I$ is known then

$${}^{A}I = {}^{A}_{B}H.{}^{B}I.{}^{A}_{B}H^{T} \tag{2.106}$$

where ${}^{A}_{B}H$ is the transformation matrix from {B} to {A}.

The reference frame associated with each rigid body must be set to in a way that the products of inertia become null. The axes of that frame are named *primary axes* of the rigid body. The eigenvalues of the inertia tensor are the so-called rigid body *primary moments of inertia.* There are some systematic methods to compute the primary axis of inertia of any rigid body [6,24].

Any rigid body plane of symmetry is perpendicular to one primary axis.

Each symmetry axis of the rigid body is a primary axis. The plane of symmetry perpendicular to that axis is a *primary plane* associated with a degenerated primary moment of inertia.

2.7.2 Lagrange-Euler Formulation

Here we briefly introduce the *Lagrange-Euler* formulation. To use this formulation, it is required to develop equations for the robot manipulator's kinetic energy and potential energy. The kinetic energy of link (i) is given by

$$k_i = \frac{1}{2} m_i V_{C_i}^T V_{C_i} + \frac{1}{2} {}^i w_i^T . {}^{C_i} I_i {}^i w_i \tag{2.107}$$

where the first term results from the linear velocity of the center of mass of link (i), and the second term is due to the angular velocity of the same link. The robot manipulator's total kinetic energy is then given by

$$K = \sum_{i=1}^{6} k_i \tag{2.108}$$

The potential energy of link (i) may be written as

$$u_i = m_i . {}^0 g^T . {}^0 P_{C_i} + u_{ref_i} \tag{2.109}$$

where ${}^0 g$ is the gravity acceleration vector, ${}^0 P_{C_i}$ is the position vector of the center of mass of link (i) expressed in terms of frame {0} and u_{ref_i} is a constant that expresses the potential energy in terms of an arbitrary origin. The total potential energy of the robot manipulator is given by

$$U = \sum_{i=1}^{6} u_i \tag{2.110}$$

The *Lagrange* equation is then

$$L = K - U \tag{2.111}$$

where K and U are obtained form (2.100) and (2.110). It follows that the motion equations of the robot manipulator can then be obtained using the Lagrange equation

$$\tau = \frac{d}{dt} \frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} \tag{2.112}$$

where τ is the joint torque vector.

Recently [4], recursive equations based on the *Lagrange-Euler* equations have been developed. The resulting equations are computationally more efficient. Nevertheless, the recursive nature destroys the equation's structure which is a

major drawback for the design and development of new control laws, and the *Newton-Euler* recursive equations remain the most efficient.

2.7.3 D'Alembert Formulation

This is basically a *Lagrange* dynamic formulation based on the *D'Alembert* principle. As mentioned before, the *Lagrange-Euler* formulation is simple but computationally inefficient, and the *Newton-Euler* formulation is compact with a recursive non-structured nature and is computationally very efficient. To obtain a recursive and computationally efficient set based on the Lagrange mechanics, a vector representation along with the use of rotation matrices is used to develop the kinetic and potential energy equations. The same procedure used in the *Lagrange-Euler* formulation is then used to compute the motion equations. This procedure is known as *D'Alembert* formulation, and is a generalization of the *Lagrange-Euler* and *Newton-Euler* formulations [7].

2.7.4 Newton-Euler Formulation

The *Newton-Euler* formulation will be used to obtain the dynamic equations of the ABB IRB 1400 industrial robot and in the process explained in some detail. We will also compare this to the other dynamic formulations.

If the joint positions, velocities, and accelerations of the robot manipulator are known, along with the kinematics and mass distribution, then we should be able to compute the required joint moments. On the other hand, if the joint torques is known, along with the inverse kinematics and the robot mass distribution, we should be able to compute the joint positions.

The *Newton-Euler* dynamic formulation is a set of recursive equations, divided in two groups: *forward recursive equations* and *inverse recursive equations*.

Forward Recursive Equations
This set of equations is used to compute ("*propagate*") link velocities and accelerations from link to link, starting from link 1 (the first link).

Angular Acceleration Computation
Using equations (2.50) and (2.51) gives

$$^{i+1}w_{i+1} = {}^{i+1}_{i}R.{}^{i}\dot{w}_i + {}^{i+1}_{i}R.{}^{i}w_i \times \dot{\theta}_{i+1}.{}^{i+1}Z_{i+1} + \ddot{\theta}_{i+1}.{}^{i+1}Z_{i+1} \qquad (2.113)$$

for the angular acceleration of link (i+1) expressed in terms of (i+1).

Linear Acceleration Computation
Using equations (2.52) and (2.53) gives

$$^{i+1}v_{i+1} = {}^{i+1}_{i}R\left[{}^{i}\dot{w}_i \times {}^{i}P_{i+1} + {}^{i}w_i \times ({}^{i}w_i \times {}^{i}P_{i+1}) + {}^{i}\dot{v}_i\right] \quad (2.114)$$

for the linear acceleration of link (i+1) expressed in terms of (i+1).

Linear Acceleration Computation at the Link Center of Mass
Using again equations (20) and (25) results,

$$^{i}\dot{v}_{C_i} = {}^{i}\dot{w}_i \times {}^{i}P_{C_i} + {}^{i}w_i \times ({}^{i}w_i \times {}^{i}P_{C_i}) + {}^{i}\dot{v}_i \quad (2.115)$$

where $\{C_i\}$ is the reference frame associated with the center of mass of link (i), and having the same orientation of $\{i\}$.

Gravity effects
The gravity effects can be included in the above equations by making

$$^{0}\dot{v}_0 = G \quad (2.116)$$

where $G = \{g_x, g_y, g_z\}^T$ is the gravity acceleration vector with $|G| = 9{,}8062$ m/s^2. This is equivalent to consider that the robot manipulator has a linear acceleration of one G, pointing up, which produces the same effect on the robot links as the gravity acceleration.

Using the above equations (2.113)-(2.115), the *Newton* equation (2nd law) and the *Euler* equation, it's possible to compute the total force and moment at the center of mass of each link:

$$^{i+1}F_{i+1} = m_{i+1}.{}^{i+1}\dot{v}_{C_{i+1}} \quad (2.117)$$

$$^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1}.{}^{i+1}\dot{w}_{i+1} + {}^{i+1}w_{i+1} \times {}^{C_{i+1}}I_{i+1}.{}^{i+1}w_{i+1} \quad (2.118)$$

Note:
Newton Equation (2nd law) - The total force applied to a rigid body of mass m and centre of mass acceleration $\dot{v}_C$, is given by $F = m.\dot{v}_C$.

Euler Equation - Consider a rigid body of mass m, angular velocity w, and angular acceleration $\dot{w}$. The total moment N starting the body in motion is given by $N = {}^{C}I\dot{w} + w \times {}^{C}Iw$, where ${}^{C}I$ is the rigid body inertia tensor expressed in terms of the reference frame associated with the body's center of mass.

Backward Recursive Equations

This set of equations is used to compute ("*propagate*") link forces and moments from link to link, starting at the last link.

Computation of Links Forces and Moments

Taking

f_i = force applied at link (i) by link (i-1);
n_i = moment in link (i) due to link (i-1);

the force balancing on link (i) can be expressed as

$$^{i}F_i = {}^{i}f_i - {}^{i+1}_{i}R.{}^{i+1}f_{i+1} \tag{2.119}$$

and the moment balancing in the center of mass of link (i) can be expressed as

$$^{i}N_i = {}^{i}n_i - {}^{i}n_{i+1} + (-{}^{i}P_{C_i}) \times {}^{i}f_i - ({}^{i}P_{i+1} - {}^{i}P_{C_i}) \times {}^{i}f_{i+1} \tag{2.120}$$

Using (2.119) in (2.120) gives

$$^{i}N_i = {}^{i}n_i - {}^{i}_{i+1}R.{}^{i+1}n_{i+1} + {}^{-i}P_{C_i} \times {}^{i}F_i - {}^{i}P_{i+1} \times {}^{i}_{i+1}R.{}^{i}f_{i+1} \tag{2.121}$$

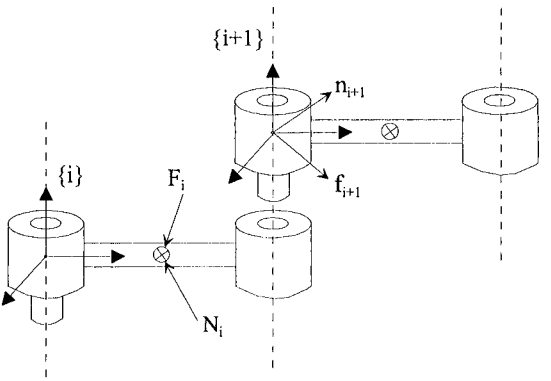


Figure 2.21 Forces and torques applied to the joints

Rewriting (2.119) and (2.121) in a way that their recursive nature becomes more evident results in

$$^{i}f_i = {}^{i}_{i+1}R.{}^{i+1}f_{i+1} + {}^{i}F_i \tag{2.122}$$

$$^{i}n_i = {}^{i}N_i + {}^{i}_{i+1}R.{}^{i+1}n_{i+1} + {}^{i}P_{C_i} \times {}^{i}F_i + {}^{i}P_{i+1} \times {}^{i}_{i+1}R.{}^{i+1}f_{i+1} \tag{2.123}$$

To obtain the joint moments we just need to project over the Z axis the already computed moment ${}^{i}n_i$, i.e.,

$$\tau_i = {}^{i}n_i^T \cdot {}^{i}Z_i \tag{2.124}$$

Contact Forces
The contact forces and moments (contact wrench) can be included in the model by putting,

$$\begin{pmatrix} {}^{N+1}f_{N+1} \\ {}^{N+1}n_{N+1} \end{pmatrix} = \text{Contact wrench} \neq 0 \tag{2.125}$$

where N is the number of degrees of freedom of the robot manipulator.

2.7.5 Dynamic Parameters

There is a number of parameters that are needed to compute the dynamic model (dynamic parameters). The minimum set of parameters is called the base dynamic parameters, and its identification can reduce significantly the computational load of the dynamic model (by 50%). If we take a closer look at the equations developed for the kinematics energy and for the potential energy of link (i), it is easy to verify that they are linear with respect to some dynamic parameters: the link mass, the six elements of the link inertia tensor, and the three components of the link's first moments of inertia. Some other dynamic parameters must also be included, namely the ones related with joint actuation. The joint torque is given by

$$\tau = \tau_m + \tau_v + \tau_f + \tau_g + \tau_\alpha + \tau_e \tag{2.126}$$

where $\tau_m = M(\theta)\ddot{\theta}$ is the torque due to the inertia of the robot manipulator, τ_v is the torque due to the centrifugal and coriolis forces, τ_f is the torque due to the friction forces, τ_g is the torque due to the gravity force, τ_α is the torque resulting from non-modeled forces and τ_e is the torque due to external contact forces.

Now, τ_m can be written as $\tau_m = \tau_{mr} + \tau_{mm}$, where τ_{mr} is the torque due to the robot manipulator inertia (not including the motor drive) and τ_{mm} is the torque due to the motor inertia itself. We may express τ_{mm} as

$$\tau_{mm} = I_m \cdot \ddot{\theta} = \begin{bmatrix} I_{m_1} & 0 & \cdots & 0 \\ 0 & I_{m_2} & \cdots & \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & I_{m_n} \end{bmatrix} \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \cdots \\ \ddot{\theta}_n \end{bmatrix} \tag{2.127}$$

where I_m is the rotor's moment of inertia and n is the number of degrees of freedom.

The friction torque may be given by

$$\tau_f = F_s.\mathrm{sgn}(\dot{\theta}) + F_v.\dot{\theta} = \begin{bmatrix} F_{s_1} & 0 & \dots & 0 \\ 0 & F_{s_2} & \dots & \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & F_{s_n} \end{bmatrix} .\mathrm{sgn} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dots \\ \dot{\theta}_n \end{bmatrix} + \begin{bmatrix} F_{v_1} & 0 & \dots & 0 \\ 0 & F_{v_2} & \dots & \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & F_{v_n} \end{bmatrix} . \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dots \\ \dot{\theta}_n \end{bmatrix} \tag{2.128}$$

where the first term refers to the coulomb friction and the second to the viscous friction.

In conclusion, I_{m_i}, F_{s_i} and F_{v_i} are also dynamic parameters to take into account, i.e., the all number of dynamic parameters is thirteen:

$$\pi = \left(I_{xx_i} \quad I_{yy_i} \quad I_{zz_i} \quad I_{xy_i} \quad I_{xz_i} \quad I_{yz_i} \quad m_i r_{ix} \quad m_i r_{iy} \quad m_i r_{iz} \quad m_i \quad I_{m_i} \quad F_{s_i} \quad F_{v_i}\right) \tag{2.129}$$

The basic Newton-Euler recursive algorithm resumed in the following form:

Forward recursive equations
Initial conditions

${}^0w_0 = 0$; ${}^0\dot{w}_0 = 0$; ${}^0\dot{v}_0 = {}^0\ddot{p}_0 = {}^0\hat{g} = (0 \quad 0 \quad g)^T$, with g = - 9,8062 m/s^2.

For i = 1 to 5,

${}^{i+1}w_{i+1} = {}^{i+1}_iR.{}^iw_i + {}^{i+1}_iR.{}^iw_i \times \dot{\theta}_{i+1}.{}^{i+1}Z_{i+1} + \ddot{\theta}_{i+1}.{}^{i+1}Z_{i+1}$

${}^{i+1}v_{i+1} = {}^{i+1}_iR\left[{}^i\dot{w}_i \times {}^iP_{i+1} + {}^iw_i \times ({}^iw_i \times {}^iP_{i+1}) + {}^i\dot{v}_i\right]$

${}^i\dot{v}_{C_i} = {}^i\dot{w}_i \times {}^iP_{C_i} + {}^iw_i \times ({}^iw_i \times {}^iP_{C_i}) + {}^i\dot{v}_i$

${}^{i+1}F_{i+1} = m_{i+1}.{}^{i+1}\dot{v}_{C_{i+1}}$

${}^{i+1}N_{i+1} = {}^{C_{i+1}}I_{i+1}.{}^{i+1}\dot{w}_{i+1} + {}^{i+1}w_{i+1} \times {}^{C_{i+1}}I_{i+1}.{}^{i+1}w_{i+1}$

Backward recursive equations
Initial conditions

End-effector wrench = $\begin{pmatrix} {}^{N+1}f_{N+1} \\ {}^{N+1}n_{N+1} \end{pmatrix}$

For i = 6 to 1,

${}^if_i = {}^i_{i+1}R.{}^{i+1}f_{i+1} + {}^iF_i$

$${}^{i}n_i = {}^{i}N_i + {}^{i}_{i+1}R \cdot {}^{i+1}n_{i+1} + {}^{i}P_{C_i} \times {}^{i}F_i + {}^{i}P_{i+1} \times {}^{i}_{i+1}R \cdot {}^{i+1}f_{i+1}$$

$$\tau_i = {}^{i}n_i^T \cdot {}^{i}Z_i$$

The generalized force at joint (i) is then

$$\mu_i = {}^{i}n_i^T \cdot {}^{i}Z_i + I_{m_i}\ddot{\theta}_i + F_{s_i}\,\mathrm{sgn}(\dot{\theta}_i) + F_{v_i}\dot{\theta}_i + \tau_{v_i} \tag{2.130}$$

2.8 Matlab Examples

Taking advantage of the preceding discussion, namely the application to the specific manipulator used for demonstration, along with the particularities of *Matlab*, a few functions were built to show how the above presented results could be used to simulate and operate a robot from *Matlab*. The functionality of this collection of functions is extended by the developments presented in chapter's 3 and 4 of this book, which enable the user to command the real robot from the *Matlab* shell.

Several functions were implemented to compute the direct and inverse kinematics, any rotation or transformation matrix, the jacobian (using the method presented here or the differential method presented in [25]), the DLS jacobian, trajectories in the Cartesian or in the joint space, simulate the operation of the robot, etc. The functions developed are related with the robot used for demonstration (ABB IRB1400), i.e., there was no effort to make them compatible with any other type of industrial robot. Consequently, the presented functions were optimized for anthropomorphic robots with a spherical wrist, with the direct and inverse kinematics obtained symbolically using *Matlab* and further optimized.

To demonstrate the functionality of the developed functions, a few examples will be given below.

Jacobian
Functions: jacobian.m and jacobdls.m
Parameters: jacobian (dh, q, type) and jacobdls(dh, q, type) where,
'dh' - *Denavit-Hartenberg* parameters od the robot
'q' - vector or array of vectors containing the joint angles representing a configuration or a sequence of configurations of the robot
'type' - method used to compute the jacobian:
'a' - returns the base jacobian and the *end-effector* jabobian of using differential method presented in [25]
'b' - returns the base jacobian using the same method [25]
'e' - returns the base jacobian using the kinematics developed in this book
'd' - returns the both jacobians using the kinematics developed in this book
'f' - returns the *end-effector* jacobian using the kinematics developed in this book

Figure 2.22 shows the utilization of the above functions to compute the jacobian of the robot for the configuration $q_1 = (0\ 0\ 0\ 0\ 0\ 0)$.

```
» flops(0)
» J=jacobian(dh,q1,'e')

J =

     0  -720  -120     0     0     0
   955     0     0     0     0     0
     0   805   805     0    85     0
     0     0     0     1     0     1
     0    -1    -1     0    -1     0
     1     0     0     0     0     0

» flops

ans =

   188

» flops(0)
» J=jacobian(dh,q1,'b')

J =

    0.0000 -720.0000 -120.0000         0    0.0000         0
  955.0000    0.0000    0.0000         0    0.0000         0
    0.0000  805.0000  805.0000         0   85.0000         0
         0         0         0    1.0000         0    1.0000
    0.0000   -1.0000   -1.0000   -0.0000   -1.0000   -0.0000
    1.0000    0.0000    0.0000   -0.0000    0.0000   -0.0000

» flops

ans =

        3412
```

Figure 2.22 Computing the jacobian: note the reduction of floating point operations when the optimized kinematics is used.

Inverse Kinematics
Function: irb14ink.m
Parameters: irb14ink(dh, t06, quad) where,
'dh' - *Denavit-Hartenberg* parameters of the robot
't06' - Transformation matrix T_6^0 that describes the position/orientation of the terminal element in terms of the base frame
'quad' – indication of the working quadrant. If nothing is given, the routine admits that the working quadrant is equal to the quadrant of θ_1

Figure 2.23 shows the function running applied to a singular configuration with indication of the working quadrant.

```
» qc

qc =

    0.7854    1.0472    0.7854         0         0         0

» t06=irb14mtr(qc',0,0,6)

t06 =

  1.0e+003 *

   -0.0007    0.0007   -0.0002   -0.4906
   -0.0007   -0.0007   -0.0002   -0.4906
   -0.0003    0.0000    0.0010    1.5215
         0         0         0    0.0010

» irb14ink(dh,t06,'q1')
Singular Point -> sin(q5)=0
Resolving Singular Point ...

ans =

   45.0000   45.0000   45.0000    0.7854    0.7854    0.7854
   60.0000   60.0000   60.0000    1.0472    1.0472    1.0472
   45.0000   45.0000   45.0000    0.7854    0.7854    0.7854
         0  -90.0000   90.0000         0   -1.5708    1.5708
         0         0         0         0         0         0
         0   90.0000  -90.0000         0    1.5708   -1.5708
   57.2958   57.2958   57.2958    1.0000    1.0000    1.0000
```

Figure 2.23 Computing the inverse kinematics (initial robot configuration expressed in radians)

2.9 Robot Control Systems

Robot control systems (Figure 2.24) are electronic programmable systems responsible for moving and controlling the robot manipulator, providing also the means to interface with the environment and the necessary mechanisms to interface with regular and advanced users or operators.

In this section, a brief overview of actual industrial robot control systems is presented, pointing out the important factors that must be addressed either by the advanced user (programmer or system integrator) or by the simple operator. Although the discussion is kept general and valid for any robot controller, a particular robot control system (the ABB IRC5 robot controller [26]) will be used for demonstration.

The robot controller has some important tasks it should perform in order to move and control the robot manipulator, provide means for inter-controller and computer communications, enable a sensor interface, and offer the necessary mechanisms and features that allow robot programming, a robot-user interface and program execution.

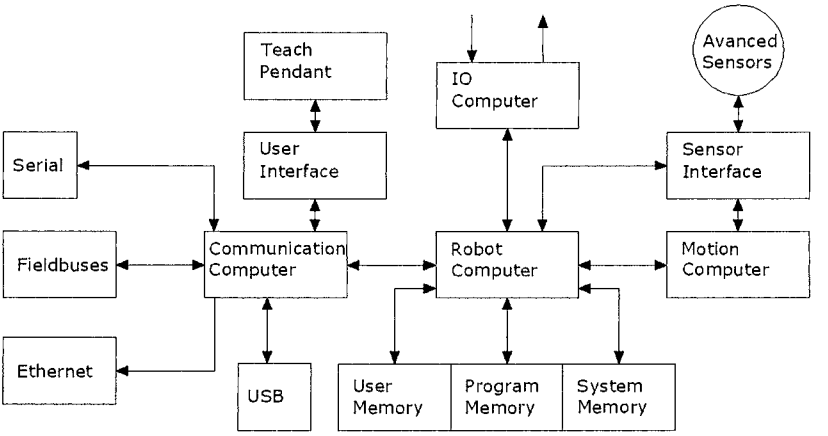


Figure 2.24 Basic architecture of a robot control system

2.9.1 Drive the motors to move the TCP and coordinate the motion for useful work

Motion control involves several different tasks, as already mentioned and resumed in Figure 2.25.

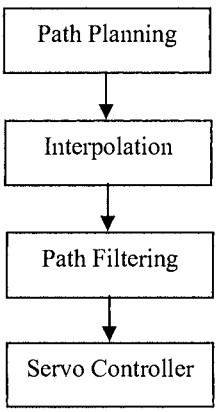


Figure 2.25 Basic tasks involved in motion control

The path planner's basic task is to prepare the robot's path and feed the relevant data to the path interpolator. Moving a robot means specifying an origin position/orientation $\{T_i\}$ and a final position/orientation $\{T_f\}$ of the robot's TCP

(*tool center point*). The path interpolator takes the planner data and computes the intermediate points in each interpolation interval, using the specified velocity and acceleration. The outputs of the interpolator are the basic inputs for the servo loops, i.e., they constitute the target points (references) that must be achieved by the servo controllers. The data from the interpolator is filtered by the path filter, before being passed to the servo controllers, in order to provide smoother accelerations/decelerations and keep the motor torques in the range of the servo-motor.

A complete definition of the motion parameters, including velocities and accelerations, is also necessary. Sometimes it is necessary to define intermediate position/orientation points (also called "*via points*") between the initial and final configurations. This procedure will better define the requirements and contribute for the final path. Furthermore, to obtain smooth paths the path planner must be a continuous function, with a continuous first derivative and hopefully also a continuous second derivative [1]. For example, the path generator can be implemented by a 5^{th} order polynomial. The use of a high-order polynomial here is motivated by the fact that a quintic polynomial is needed to be able to specify the position, velocity, and acceleration at the beginning and end of each path segment.

Considering a 5^{th} order polynomial in the form

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{2.131}$$

with the following constraints

$$\begin{aligned}
\theta_0 &= a_0 \\
\theta_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 \\
\dot{\theta}_0 &= a_1 \\
\dot{\theta}_f &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \\
\ddot{\theta}_0 &= 2a_2 \\
\ddot{\theta}_f &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3
\end{aligned} \tag{2.132}$$

Results in a linear system of six equations with six unknowns whose solutions are

$$\begin{aligned}
a_1 &= \theta_0 \\
a_1 &= \dot{\theta}_0 \\
a_2 &= \frac{\ddot{\theta}_0}{2} \\
a_3 &= \frac{20\theta_f - 20\theta_0 - (8\dot{\theta}_f + 120\dot{\theta}_0)t_f - (3\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^3}
\end{aligned}$$

$$a_4 = \frac{30\theta_0 - 30\theta_f + (14\dot{\theta}_f + 16\dot{\theta}_0)t_f - (3\ddot{\theta}_0 - 2\ddot{\theta}_f)t_f^2}{2t_f^4}$$

$$a_5 = \frac{12\theta_f - 12\theta_0 - (6\dot{\theta}_f + 6\dot{\theta}_0)t_f - (\ddot{\theta}_0 - \ddot{\theta}_f)t_f^2}{2t_f^5} \qquad (2.133)$$

There are several methods in the literature to compute smooth paths that pass to a given set of "*via points*" [27, 28]. Nevertheless, the function presented above gives a good indication and can be used for that objective, running the function between the intermediate points.

The following *Matlab* functions (Figure 2.26) calculate the robot's trajectory in the joint space using the 5th order polynomial presented above. As already mentioned, with this trajectory planner it is possible to compute the trajectory between two configurations, defining the initial and final velocities and accelerations. The trajectory is represented using a small function that animates the motion of the robot.

Trajectory generation and robot animation
Funtions: irb14trj.m and irb14plt.m
Parameters: [qt, qdt, qddt] = irb14trj(q0, q1, nt, qd0, qd1, qdd0, qdd1) and irb14plt(dh, q, opt, number, azm, elv, vgax, vgay) where,
'q0' – initial position
'q1' – final position
'nt' – number of intermediate points of the trajectory to obtain
'qd0' and 'qd1' – initial and final values of the velocity
'qdd0' and 'qdd1' – initial and final values of the acceleration
'dh' – *Denavit-Hartenberg* parameters of the robot
'q' – matrix holding the computed trajctory
'opt' – type of representation of the motion

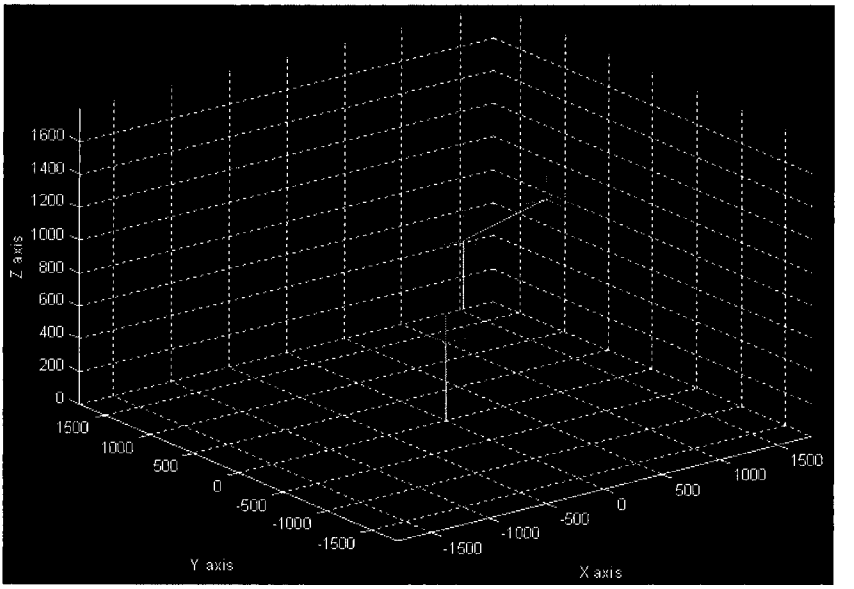


Figure 2.26 Robot's animation using the obtained trajectory

2.10 Servo Control

The servo controllers utilize the data from the path planner and interpolator, properly filtered, to drive the robot manipulator axis. As already mentioned the dynamics of the robot is very complex with a huge number of effects, forces and moments to account for, which puts a considerable challenge to the task of controlling a servo-motor. A detailed and complete description of a servo-controller, namely about the control algorithms and circuitry used, is out of the scope of this book, but a brief overview will be given. Generally, the control loop of an industrial robot joint (or axis) has the components presented in Figure 2.27.

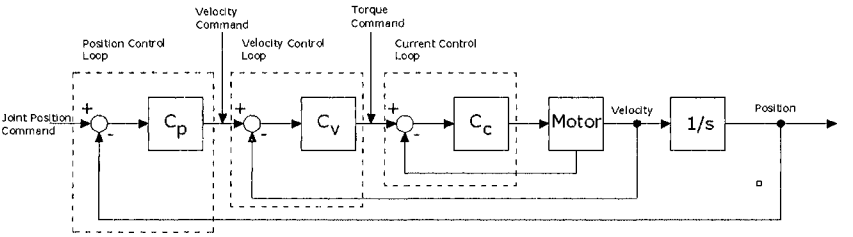


Figure 2.27 Typical robot joint control loop

A brief overview of the AC motors used with industrial robots was already presented, and a typical current control loop was also already sketched in Figure 2.20. Basically, the current control loop implements a PI (proportional and integral) controller [29], having the I component of the controller (Cc) with the objective of eliminating the steady-state error and achieving the best possible control. The velocity control loop is built around the current control loop and also uses a PI controller (Cv).

Finally, around both of the previous controllers there is the position control loop. This controller takes the position commands as input, generates an error signal by subtracting the actual position (obtained from the joint position sensors) from the commanded reference, and generates the control signal using some selected control law (Cp). Typically, the position controller is a simple proportional controller, since the objective is to obtain a good responsive control of the motor position to follow the desired joint command with zero steady-state error and zero overshoot. And that objective is obtained with the combined effect of the position (generally a P controller), velocity (generally a PI controller), and current (generally a PI controller) control loops.

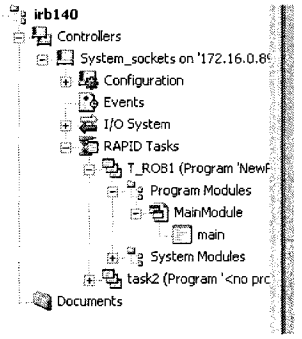
2.11 IO Control

One of the most basic things that a robot control system must do is to implement PLC-like functions. Robots are used in manufacturing cells where digital/analog IO and logic controllers govern the way things happen, namely controlling the systems responsible for material handling, transportation, detection, etc.. To interface with those systems, the robot controller needs to "speak" the same language and act as a logic controller, or at least have the same functionality available. Consequently, the robot controller must be able to:

1. Accommodate digital IO signals with variable and configurable electric levels. The robot must be able to read from digital input lines (with different electric levels) and implement basic logic functions on the obtained data: block reading, logic functions, shifting, counters, timers, edge detection, etc. The robot controller must also be able to act on digital IO outputs changing their state (ON/OFF), applying timed pulses, etc.
2. Accommodate analog IO signals. The robot must be able to read from analog inputs, providing the necessary electronic circuits for multiplexing and analog-to-digital conversion, the mathematical functions to handle the results, and the necessary circuits and digital-to-analog converters to act on analog output signals.
3. Implement IO manipulating functions.

The robot controller programming language must implement advanced mathematical functions, and data structures, that can be used within the robot's

program to enable the user to coordinate the robot's motion with IO actions (Figure 2.28), like reading IO information or acting on IO lines (open/close grippers, regulate pressure of pneumatic actuators, regulate the velocity of external motors driven by power inverters or external servo controllers, start/stop equipment, etc.)




```
  decision1:=123;
ENDIF
IF decision1 = 96 THEN
  MoveJ p5, v200, z50, tool0;
  decision1:=123;
ENDIF
IF decision1 = 201 THEN
   SetDO DO07,1;
   MoveJ Offs(pick,0,0,100), v300, fine, tool0;
   MoveL pick, v50, fine, tool0;
   SetDO DO07,0;
   WaitTime 2;
   MoveL Offs(pick,0,0,100), v50, z20, tool0;
   MoveJ pick1, v300, z20, tool0;
   decision1:=123;
ENDIF
```

Figure 2.28 Part of a robot program written in RAPID (*ABB Robotics* programming language)

2.12 Communication

Robots are to used in networks with other robots and computers organized into manufacturing cells that also connect to each other constituting manufacturing lines. This type of manufacturing organization corresponds to one of the most recent developments in the area of industrial automation, i.e., the concept of *flexible manufacturing systems* (FMS). These are highly computerized systems composed by several types of equipment, usually connected through a local area network (local network using MAP[13] protocols [30]) under some hierarchical *computer integrated manufacturing* (CIM) structure [31-33]. The available factory (*shop floor*) equipment is organized into *flexible manufacturing cells* (FMCs) with transportation devices connecting the FMCs. In some cases, functionally related FMCs are organized into *flexible manufacturing lines* (FMLs). Each FML may include several FMCs with different or equal basic capabilities. The organization proposed in Figure 2.29 is a hierarchical structure [33,34] where each FMC has its own controller. Therefore, if the manufacturing process is conveniently organized as a FML, then several controllers will exist on the shop floor level, e.g., one controller for each FML. With this setup, an intelligent and distributed job dispatching and awarding may be implemented, taking advantage of the installed industrial network [33,35-37].

---

[13] *Manufacturing automation protocol* (MAP).

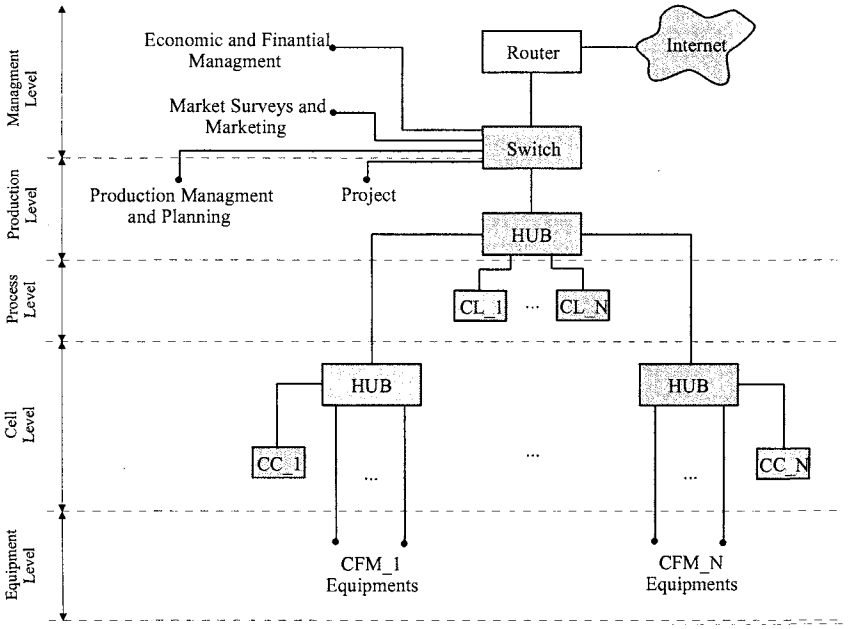


Figure 2.29 Typical CIM hierarchical organization

The best characteristic of an FMC is its flexibility, i.e., its adaptability to new manufacturing requirements that can go from a modified product to a completely new product. The flexibility results from the fact that FMC equipment is programmable and easily reconfigured: that is the case of industrial robot manipulators, mobile robots for parts handling and transportation, *programmable and logic controllers* (PLC), CNC machines, vision systems, conveyors, etc.

Considering the communication between commanding and supervising computers and the robot controllers, and even the communication between robot controllers itself, it is usually supported through a TCP/IP *Ethernet* based network. The functions associated with this type of communication include the exchange of files and programs, the execution of remote operations like backup and system maintenance, etc. In many advanced applications, this network is also used to command and supervise each manufacturing cell operation, with several levels of functionality depending on the type of access: operator access, supervisor access, or information retrieval access from the production planning levels of the network. These types of advanced features will be extensively explored in this book.

Many manufacturers offer robot services through this type of network to support these advanced applications, in the form of RPC servers [38], TCP/IP socket servers [26], or UDP datagram servers [39]. These servers and associated services can be used by the system developer/integrator to provide functionality to the user through the application.

Furthermore, the communication links between the controller and the manufacturing cell can be as follows:

1. Computer network – to interface with commanding and supervising computers, from several levels of the network
2. Fieldbuses – to interface with other robot controllers, but also with PLCs and other cell equipment commanded by programmable controllers. The most common options are *DeviceNet*, *ProfiBus*, *Ethernet IP*, etc. Several robot controllers also use a fieldbus network (*CAN* or *DeviceNet*, for example) to connect some of its internal components (the drive boards to the main computer, etc.)
3. Serial IO – to interface with sensors, or with several types of IO equipment or process equipment like welding power sources, to interface locally with a computer or laptop using a point-to-point occasional connection, and so on

2.13 Sensor Interface

Interfacing advanced sensors is a fundamental aspect of any robot control system. In fact, to successfully perform several actual industrial tasks, the robots need special sensors that could be used to help them get the relevant information and use it efficiently through the process. Many of these sensors require high-performance, non-perturbed communication links, and/or need to interface directly to the path planners and motion controllers so that the robot can be guided and instructed in real-time. Consequently, the robot controllers should provide special interfaces for these types of sensors, at least for the most common ones, which can be programmed and explored by the advanced user.

2.13.1 Interfacing Laser 3D Sensor for Seam Tracking

Good examples are the laser sensors used in robotic welding for seam finding and tracking during the welding operation. These types of sensors provide signals (analog or through high-speed digital interfaces) that can be used to guide the robot during the welding operation. These sensors work in a simple way, based on the principle of laser triangulation. A low power laser source is used to generate a laser beam that is projected onto the surface of the joint to weld. The reflected light is picked up by a lens that feeds the imaging system, composed usually of a CCD or CMOS sensor. The laser-reflected signals are extracted using filters and image processing software, which is a simple task since the laser signal has a very precise wave length and power (Figure 2.30).

In fact, these laser cameras and related processing hardware and software, with some customization to the selected application, are useful for evaluating most of the geometric parameters other than the mentioned joint detection and seam

tracking features. Since they are available with powerful APIs for general use, with standard interfaces for robot controllers and current computer hardware, these types of sensors constitute a powerful tool for robotic welding.

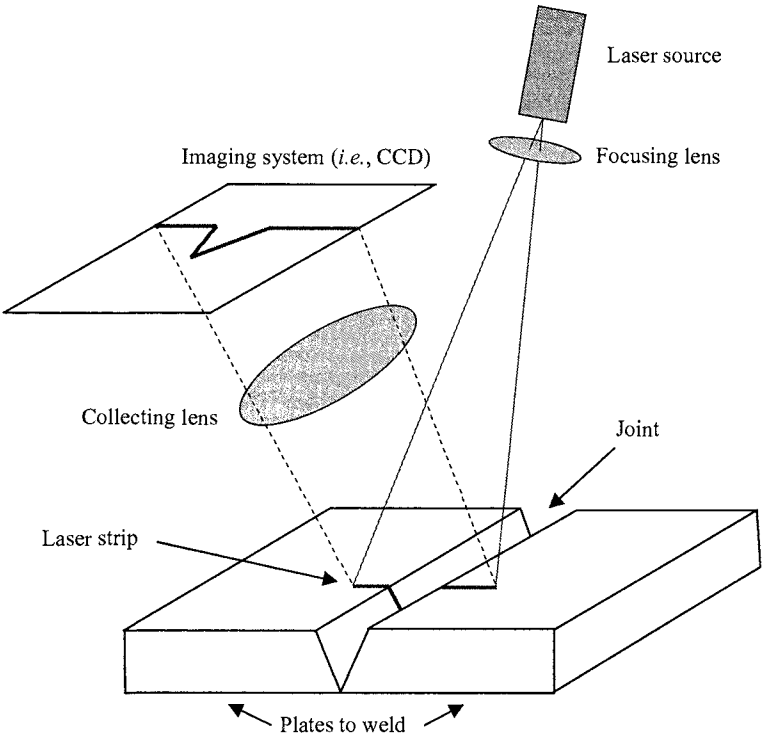


Figure 2. 30 Explanation of the laser vision principle

Basically, the outputs obtained from these sensors are position accommodations, or position corrections, that should be sent to the robot controller to adapt the current motion. They can also monitor certain variables and provide the means to generate interrupts in the robot controller in order to respond to significant variable changes. For example, the seam volume or the welding gap can be monitored by this sensor. When changes are detected, the corresponding events can be used to trigger an internal interrupt that will adapt the welding parameters (voltage, wire feed and velocity) accordingly. For example, the following would be the procedure to adapt the welding parameters in function of the measured welding gap:

Variables
Matrix Numeric Adapted_voltage = {1, 1.1, 1.2, 1.4, 1.6, 2, 2.2, ...};
Matrix Numeric Adapted_wire_feed = {2, 2.2, 2.4, 2.6, 2.8, 3, 3.2, ...};
Matrix Numeric Adapted_velocity = {10, 12, 14, 16, 18, 20, 22, ...};
Numeric gap_value;

```
   Numeric index;

Program
   Set Interrupt 1 when gap_value changes;
   Start Welding, tracking;
   When target point achieved
      Stop welding, tracking;
   EndWhen
EndProgram

Interrup Service Routine
   index = scale(gap_value);
   voltage = adapted_voltage(index);
   wire_feed = adapted_wire_feed(index);
   velocity = adapted_velocity(index);
   refresh welding parameters;
EndRoutine
```

The position of the sensor can also be read and used to accommodate the position references sent to the motion controller, guiding in this way the robot's motion.

The next example shows how to interface other type of intelligent sensors for which there is no special interface at the robot controller.

2.13.2 Interfacing a Force/Torque Sensor

As already mentioned, robot manipulators are good examples of equipment for *flexible manufacturing systems*, due to their flexibility. In fact, flexibility is the major reason for robot utilization and popularity in actual manufacturing plants. In this framework, the majority of the robot's tasks require contact with the surrounding environment, i.e., in the process of fulfilling the task, the robot tool interacts physically with the working objects and surfaces. That interaction generates contact forces that should be controlled in a way to finish the task correctly, not damaging the robot tools and working objects. Those contact forces depend on the stiffness of the tool and working objects/surfaces and should be properly controlled. The option for a particular control technique depends on identifying if [40]:

1. The contact forces should be controlled to achieve task success, but are sufficient to keep them inside some safety domain: *passive force control* [40].
2. The contact forces should be controlled because they contribute directly to the success of the task: *active force control* [40-53].

In the first case, contact forces are an undesirable effect of the task and it is generally sufficient to keep them inside some safety domain. They are not necessary for the task, so usually the strategy is adding flexibility to the *end-effector* with the object of damping all the possible impacts and increasing the

tolerance to positioning errors, complemented with detailed and careful planning of flying trajectories and object approach. There are many solutions in the market to add flexibility to the *end-effector*, and in fact this is currently the standard approach in industry.

In the second case, the contact forces are necessary to finish the task correctly, i.e., controlling the contact forces to make them assume some particular value or, more generally, to follow some force profile.

For industrial robotics applications, force/torque sensors are usually placed near the working tool, generally in the manipulator wrist. This means that the sensor must be reasonably small, built in several sizes to adapt to different robot bolt patterns and load capacities, and mechanically resistant. Considering these restrictions, it is easy to understand why measuring the strain imposed on a selected strain gauge material, just by reading the voltage across the resistance of the material, is still the most used sensing technique.

There are several ways and materials to design sensing gauges, metal wire, metal-foil and semiconductor gauges being the most common. From those, the metal-foil gauges show some interesting features. The strain induced change in resistance is due to length and sectional area changes as well as a small piezo-resistive effect. With the developments in etching processes, metal-foil gauges became a very interesting possibility. They are manufactured in very thin foils (less than 10 μm), with sizes down to 200 μm, etched by photographic methods. Consequently, there are virtually no limits to the variety of possible geometries. This gives greater flexibility to design geometries, but also to the type of stressing at the surface of the elastic material component where the gauge will be attached. Metal-foil gauges have very high linearity, with very low transverse sensibility (less than 0.3%), and great dynamic range. Also, their thermal characteristics are better than their semiconductor and metal-wire counterparts. All these arguments explain why metal-foil gauges are ideal for force/torque sensing elements. Force/torque sensors manufactured by JR3 (the ones we use in this book) use metal-foil gauges bound to elastic rings as sensing elements, which explain their superior behavior. Figure 2.31 shows the composition of these sensors.

The sensing part. It is composed of elastic rings at the outer perimeter between the mounting plates. The monolithic design eliminates hysterisis that would occur from slippage at highly stressed internal joints. The use of elastic rings produces a very stiff device, resulting in minimal deflection under load and better performance at higher frequencies. The rings and their strain gauges are positioned so that the local strain measures can be used to deduce the forces and moments, in all cartesian directions (X, Y, Z), passing through the sensor. The internal cavity between the mounting plates contains the front-end electronics where signals are amplified, digitized, and transmitted to the host receiver board. If the amplification and digitization electronics are inside the sensor, preferable for noisy or industrial environments, there is no analog signal being transmitted and high sampling rates can be achieved (8Khz).

Table 2.3 Functions available in the MATJR3PCI *Matlab* Mex file

Functions	Brief description
init_jr3	This function opens a handle to the JR3PCI driver, checks memory, and downloads DSP code to the board.
read	Reads from a receiver board memory address.
write	Writes to a receiver board memory address.
system_warnings	Reads system saturation warnings (board memory address WARNINGS).
system_errors	Reads system errors (board memory address ERRORS).
command	Commands JR3 receiver board.
get_threshold_status	Gets the value of the threshold bits (board address THRESHOLD).
reset_threshold	Resets the threshold bits.
read_ftdata	Reads force/torque data from receiver board.
set_transforms	Sets a new transformation definition.
use_transforms	Selects the transformation to use.
read_offsets	Read offsets in use.
set_offsets	Sets actual offsets, using the current offset index.
change_offset_num	Changes actual offset index (num).
reset_offsets	Sets actual offsets to the current values read from FILTER_2.
use_offset	Changes actual offsets to the one defined.
peak_data	Sets address to watch for peaks.
peak_data_reset	Sets address to watch for peaks and resets internal values to current data.
read_peaks	Reads current peak values.
bit_set	Sets bits on defined bit-map.
set_full_scales	Sets JR3 Full_Scales.
get_full_scales	Reads actual full_scales.
get_recommended_full_scales	Reads recommended full_scales.
sensor_info	Reads information from the sensor and from the receiver board. Use this function to test your setup.

Note: all these functions address a specific sensor, even if a multi-channel board is used.

DSP receiver board. Based on the same basic architecture, several interfaces can be used. If the issue is high access rates, then fast IO buses must be used and a shared memory mechanism must be implemented to exchange data and program the sensor. JR3 offers several interface buses like VME, PCI (up to four channels per board), CPCI (also up to four channels) and ISA. The receiver boards are basically DSP boards that implement digital filters and dispose sensor information to users. Also they parameterize readings (offsets, full scales, geometrical transformations, etc.) and implement a few interesting functions such as maximum

and minimum values (peaks) and, warning and error bits, etc. A full description of these functions can be found in [54], and a brief summary can be found in Table 2.3.

Interface software and drivers. For *Win32*-based operating systems, we developed a complete set of tools that can be used to build applications using force/torque sensors. These tools include kernel drivers designed for *Win32* operating systems, i.e., *Windows*. Basically, when we want to use some kind of equipment from a computer we need to write code and define data structures to handle all its functionality. We can then pack the software into libraries, which are not easy to distribute being language dependant, or build a software control using one of the several standard languages available. Having in mind that force/torque sensors can be used by persons with different programming capabilities, and from several types of programming languages and environments, the collection of functions that access the sensor capabilities are offered in several packages: C++ Library, *ActiveX* software component, *Matlab* toolbox and *LabView Virtual Instruments* [55].

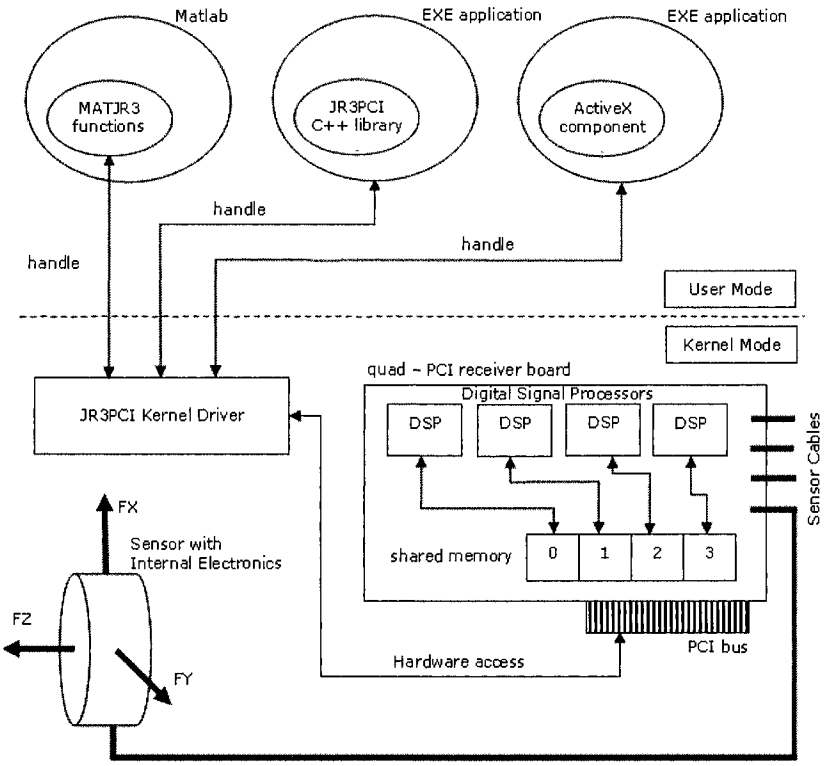


Figure 2.31 Force/torque sensor overview (using PCI receiver board)

With this organization, the sensor works like a server, offering a collection of services to the advanced user, who can use the available programming tools cited above to tailor the sensor behavior. The next section demonstrates the sensor capabilities using the popular application *Matlab*.

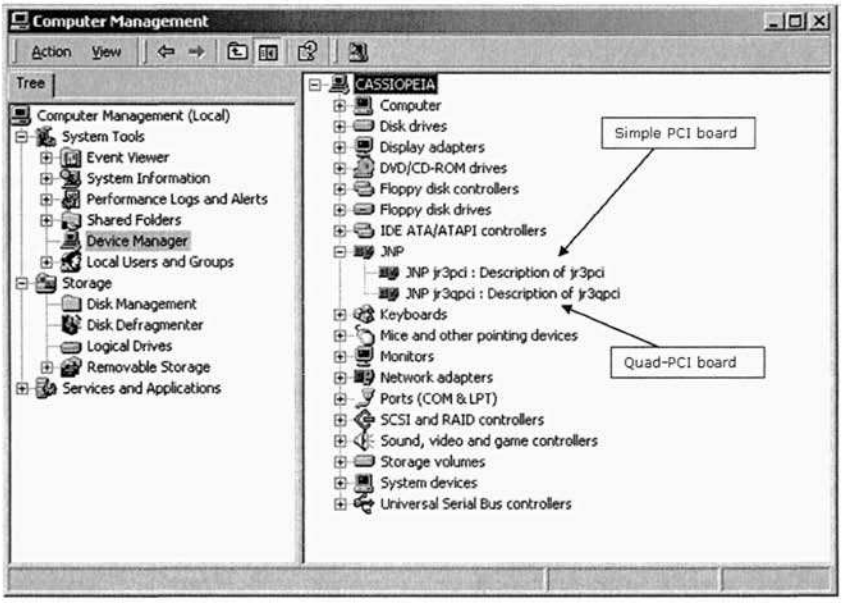


Figure 2.32 Boards reported by Windows device manager

2.13.2.1 Using a Force/Torque Sensor

There are several applications of force/torque sensors, but generally a user just wants to install the sensor on his computer (after installing the sensing part on the mechanical system he is using), and then be able to parameterize it and get the sensor readings at selected rate from within the selected environment he chose to use. The basic software [54] was prepared to be used with virtually any application or programming language under *Win32* operating systems, by any type of user: from computer experts to regular users. Here we use two different environments to explore the sensor capabilities. In this section, *Matlab* is used. *Matlab* is a widely used software environment for research and teaching applications on robotics and automation, mainly because it is a powerful linear algebra tool, with a very good collection of toolboxes that extend its basic functionality, and because it is an interactive open environment. So, it is really a good environment to demonstrate how to use this type of intelligent sensor.

From all the available receiver board models, the quad-PCI receiver model was used. This board is capable of handling four force/torque sensors at the same time on a single PCI slot. It will be used step-by-step.

After having the board installed and correctly reported by the operating system (Figure 2.32), with sensor cables attached, the user is ready to start using the sensor. The first thing to do is open a handle to the sensor receiver board, check if the board is OK, and download the DSP code to the receiver's board program memory.

The command is

>> matjr3pci('init_jr3',vendor_ID, device_ID, n_board, n_proc,download);

where *vendor_ID* and *device_ID* are the PCI ID's of the selected board, *n_board* is the board number (there can be several in the PCI bus), *n_proc* is the number of DSP units in the board, and *download* specifies if the user wants to download (1) the DSP code to the program memory or not (0). Nevertheless, DSP code must be downloaded once after each computer power-up, but after that the command can be used simply to open a handle to the board. The command returns zero if successful, or an error code [45]. Consequently, to a quad-PCI board, the command with DSP code download should be:

>> matjr3pci('init_jr3', 0x1762, 0x3114, 0, 4, 1);

or without download:

>> matjr3pci('init_jr3', 0x1762, 0x3114, 0, 4, 0);

If the return value is zero (0) then the user can start using the sensor, otherwise the user must solve the problem reported by the software (error code).

The first command could be a query to the system to find out what sensor is attached to each channel. The command is

>> matjr3pci('sensor_info', 2);

to get information about the force/torque sensor handled by DSP number 2. The returned information includes model and serial numbers, software version, number of ADC bits, etc.

To read offsets from the force/torque sensor handled by DSP number zero (remember we are using a board with 4 DSP: numbered from 0 to 3),

>> offset_matrix = matjr3pci('read_offsets', 0);

To set offsets of the force/torque sensor handled by DSP number 2,

>> matjr3pci('set_offsets',matrix_off, 2);

where *matrix_off* is a matrix with the offset values.

To reset offsets,

>> matjr3pci('reset_offsets', n_dsp);

where *n_dsp* is the DSP number. With this function, the offsets are zeroed using the actual values reported by FILTER_2 [56].

The offsets are stored in the memory available for each DSP. It is possible to store 16 independent tables of offsets for each DSP. Consequently, before any of the previous operations, the user should define the table currently in use. If the definition is not performed, all operations are referent to the actual table. To set a table for offset reading the command is,

>> matjr3pci('change_offset_num, 12, 1);

to specify that all subsequent offset operations for the sensor handled by DSP number 1 are to be addressed to Table 12. Table 12 is also used on any subsequent force/torque reading for that sensor.

To specify a table for actual force/torque readings the command is,

>> matjr3pci('use_offset, 10, 2);

where table 10 was selected for sensor handled by DSP number 2.

Another important operation on this type of sensor is setting the full-scales to properly scale the readings. This operation is similar to the operations of setting and reading offsets, so it will not be mentioned explicitly.

Each DSP has an address space [56]. To read, write, and issue commands relative to those address spaces the user should use the *read*, *write,* and *command_jr3* commands. For example, to read the serial number (address 0x00f8 of each DSP address space) of the force/torque sensor attached to DSP number 2 the command is,

>> serial_2 = matjr3pci('read_jr3', 248, 2);

Finally, to read data from any sensor the command is,

>> ft_data = matjr3pci('read_ftdata', n_filter, n_dsp);

where *n_filter* is the filter number (from 0 to 6, where 0 means unfiltered data), and *n_dsp* is the DSP number.

The collection of functions available from this *Matlab* toolbox can be found in [54] and the correspondent functions of the C++ library or *ActiveX* control can be found in [57]. The same basic function prototypes have been kept between all the

software packages, which makes the above *Matlab* demonstration a good way to show how the other packages work (C++ library, *ActiveX* control, etc).

This example demonstrates how to interface an intelligent sensor to a computer. If the same facilities were available from the robot controller, then it would be equally easy to make the interface available directly from the controller, enabling in this way the programmer to directly use its readings to influence the robot's motion. Nevertheless, with most of the commercial robot controllers, this type of advanced access is not available or isn't accessible. Consequently, these types of sensors must be used form personal computers feeding the data to the robot using the available communication channels. This type of indirect approach slows down the possible performance, but it's an alternative way to implement the interface to the force/torque sensors.

2.14 Programming and Program Execution

Robot controllers should provide a programming language and a library of functions to enable users to explore the functionalities of the robot and of the robot's controller. Most of the manufacturers offer advanced *PASCAL-like* structured programming languages, including a language interpreter within the controller. Consequently, users can write code using any ASCII editor, download it to the controller, and run it immediately without the need for any type of file conversion. Those programming environments also offer simple debugging tools that make the process of developing software easy.

The most advanced manufacturers also offer online and offline PC-based programming tools, which enable users to develop code directly in the controller (online) using a remote PC. Alternatively, the code can be developed offline and downloaded to the controller when ready.

The Teach Pendant Unit (TPU) can also be used to program and parameterize the system. These devices are basically computer units running a local operating system (*Windows CE*, for example) that offer to several types of users the possibility to program, parameterize, and operate the robot manipulator.

The actual robot controllers are also multitasking systems, which enable the user to develop and run multiple tasks simultaneously. This allows new levels of functionality, offering new possibilities to the system developer. Using the available and common inter-task communication mechanisms, along with the ability to regulate task priorities (percentage of CPU time), it's possible to set up applications to handle all the challenges posed by the industrial manufacturing cells.

2.15 User Interface

The user interface is basically defined by the system developer, because there are a lot of possibilities. The developer can use the available communication links and the robot controller's remote servers to set up a PC interface to command and monitor the robot operation (see for example Figures 1.20 and 1.21). Alternatively, he can use the controller TPU to design the user interface. Since most of the current teach pendants are advanced computers, running powerful operating systems, the possibilities for developing advanced interfaces are enormous and flexible.

For example, the TPU that comes with the new ABB IRC5 controller [26] is a *Windows CE* system (Figure 2.33), equivalent to any portable CE based consumer device, which can be programmed remotely from a PC using common programming tools like the *Microsoft Visual Studio .NET* programming suite.

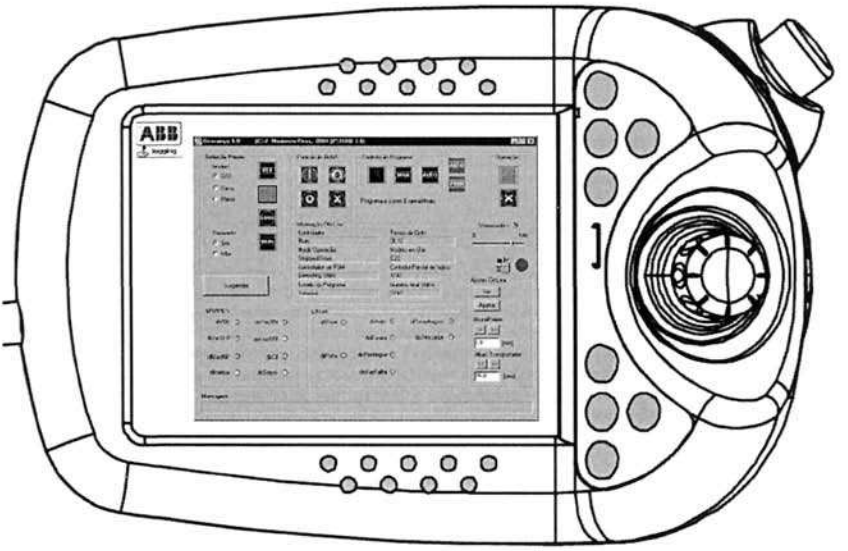


Figure 2.33 Teach Pendant Unit showing a graphical user interface

This book explores several examples that use a remote PC to implement the user interface, examples that use mainly the TPU, and examples that use both possibilities. The idea is to demonstrate that the possibilities are there and that it's up to the system developer to pick the best options for the specific application he's building.

2.16 References

[1] Craig, J.J., "Introduction to Robotics, Mechanics and Control", 2ª Edition, Addison-Wesley, 1989.

[2] Sciavicco, L., and Siciliano, B., "Modeling and Control of Robot Manipulators"-2nd Edition, McGraw-Hill, 1996.

[3] De Wit, C.C., Siciliano, B., Bastin B., "Theory of Robot Control", Springer-Verlag, London, 1996.

[4] Hollerbach, J.M., "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity", IEEE Transactions on Systems, Man and Cybernetics, Novembro de 1980.

[5] Pieper, D.L., "The Kinematics of Manipulators Under Computer Control", memo. AIM72, Stanford Artificial Intelligence Laboratory, 1968.

[6] Symon, K.R., "Mechanics", 3ª Edition, Addison-Wesley, 1971.

[7] Fu, K., Gonzalez, R., Lee, C.S.G., "Robotics: Control, Sensing, Vision and Intelligence", McGraw-Hill, 1987.

[8] Klema, V.C., Laub, A.J., "The Singular Value Decomposition: Its Computation and Some Applications", IEEE Transactions on Automatic Control, Vol. AC-25, Nº 2, April 1980.

[9] Chiaverini, S., Siciliano, B., Egeland, O., "Review of the Damped Least-Squares Inverse Kinematics with Experiments on an Industrial Robot Manipulator", IEEE Transactions on Control Systems Technology, Vol.2, Nº2, June 1994.

[10] Golub, G.H., Van Loan, C.F., "Matrix Computations", The Johns Hopkins University Press, Baltimore, Maryland, 1983.

[11] Wampler, C.W., Leifer, L.J., "Applications of Damped Least-Squares Methods to Resolved-Rate and Resolved-Acceleration Control of Manipulators", Journal of Dynamical Systems, Measurement and Control, Vol.110, January of 1988.

[12] Golub, G.H., Klema, V.C., Stewart, G.W., "Rank Degeneracy and Least Squares Problems", Department of Computer Science, Stanford University, Technical Report STAN-CS-76-559, August 1976.

[13] Maciejewski, A.A., Klein, C.A., "Numerical Filtering for the Operation of Robotic Manipulators through Kinematically Singular Configurations", Journal Robotics Systems, Vol.5, 1988.

[14] Chiaverini, S., "Estimate of the Two Smallest Singular Values of the Jacobian Matrix: Application to Damped Least-Squares Inverse Kinematics", Journal of Robotic Systems, Vol.10, Nº8, 1993.

[15] Luh, J.Y.S., Walker, M.W., Paul, R.P.C., "Resolved-Acceleration Control of Mechanical Manipulators", IEEE Transactions on Automatic Control, Vol.AC-25, 1980.

[16] Dote, Y., "Servo Motor and Motion Control using Digital Signal Processors", Texas Instruments and Prentice-Hall, 1990.

[17] Herdershot Jr., J.R., and Miller, T.J.E., "Design of Brushless Permanent-Magnet Motors", Magna Physics Publishing and Clarendon Press, Oxford, 1995.

[18] Crowder, R.M., "Electric Drives and their Controls", Clarendon Press, Oxford, 1995.

[19] Tamagawa Seiki Co. Ltd, "SmartSyn Brushless Resolvers, General Catalog", Tamagawa Seiki, Japan, 2005.

[20] ABB Robotics, "S4-IRB1400 Product Manual" - M94A, ABB Flexible Automation, 1994.

[21] Hanselman, D.C., "Techniques for Improving Resolver-to-Digital Conversion Accuracy", IEEE Transactions on Industrial Electronics, Vol.38, No. 6, December 1991.

[22] Boyes, G., "Synchro and Resolver Conversion", Analog Devices Inc. (Norwood, MA), 1980.
[23] Analog Devices, "AD2S80A Resolver to Digital Converter - Data Sheet", Data Conversion Manual, 1995.
[24] Goldstein, H., "Classical Mechanics", 2ª Edição, Adison-Wesley, 1980.
[25] Paul, Shimano and Mayer, "Differential Kinematic Control for Simple Manipulators", IEEE Trans. SMC Vol.11, n.6 Junho de 1981.
[26] ABB Robotics, "IRC5 documentation CD", ABB Robotics, 2005
[27] Deboor, C., "A Practical Guide to Splines", Springer, New-York, 1979
[28] Rogers, D., Adams, J.A., "Mathematical Elements for Computer Graphics", McGraw-Hill, 1976.
[29] Ogata, K., "Modern Control Engineering" Prentice-Hall Inc., 1970.
[30] Halsall F., "Data Communications, Computer Networks and Open Systems", Third Edition, Addison-Wesley, 1992.
[31] Kusiak A., "Modelling and Design of Flexible Manufacturing Systems", Elsevier Science Publishers, 1986.
[32] Ou-Yang C. and Lin JS., "The Development of a Hybrid Hierarchical/ Heterarchical Shop Floor Control System Applying Bidding Method in Job Dispatching", Robotics and Computer-Integrated Manufacturing, 1998;14(3):199-217.
[33] Waldner JB., "CIM, Principles of Computer Integrated Manufacturing", John Wiley & Sons, 1992.
[34] Liang GR., "A Hybrid Model of Hierarchical Control Architecture in Automated Manufacturing Systems", in Advances in Factories of the Future, CIM and Robotics, Elsevier Science Publishers, 1993:277-286.
[35] Baker AD., "Complete Manufacturing Control Using a Contract Net: A Simulation Study", Proceedings of the IEEE International Conference on Computer Integrated Manufacturing, 1988: 100-9.
[36] Shaw M., "A Distributed Scheduling Method for Computer Integrated Manufacturing; the use of Local Area Networks in Cellular Systems", International Journal on Production Research, 1987;25(9):1285-1303.
[37] Zhang Y., Kameda H. and Shimizu K., "Adaptive Bidding Load Balance Algorithms in Heterogeneous Distributed Systems", Proceedings of the IEEE Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1994:250-254.
[38] RAP, Service Protocol Definition, ABB Flexible Automation, 1996.
[39] Remote Connection Manual for the NX100 Controller, Motoman Robotics, 2005
[40] Siciliano B., Villani L., *Robot Force Control*, Kluwer Academic Publishers International Series in Engineering and Computer Science, Boston, MA, 1999
[41] Pires, JN, and Sá da Costa JMG, "A Real Time System for Position/Force Control of Mechanical Manipulators", Proceedings of the 7th International Machine Design Conference, Ankara, Turkey, 1996.
[42] De Schutter, J., and Van Brussel H., "Compliant Robot Motion I. A Formalism for Specifying Compliant Motion Tasks", The International Journal of Robotics Research, August de 1988.
[43] De Schutter, J. and Van Brussel, H., "Compliant Robot Motion II. A Control Approach Based on External Control Loops", The International Journal of Robotics Research, August, 1988.
[44] Craig JJ, and M.H Raibert MH, "A Systematic Method of Hybrid Position/Force Control of a Manipulator", IEEE Computer Software Applications Conference, November, 1979.
[45] Nilsson K., "Industrial Robot Programming", Ph.D. Thesis, Department of Automatic Control, Lund Institute of Technology, May of 1996.

[46] Hogan N., "Impedance Control: An Approach to Manipulation: Part I-Theory, Part II-Implementation, Part III-Applications", ASME Journal of Dynamic Systems, Measurement, and Control", March, 1985.
[47] Khatib O., "A unified Approach for Motion and Force Control of Robotic Manipulators: The Operational Space Formulation", IEEE Journal of Robotics and Automation, February 1987.
[48] Volpe R. and Khosla P., "A theorical and Experimantal Investigation of Explicit Force Control Strategies for Manipulators", IEEE Transactions on Automatic Control, November, 1993.
[49] Volpe R. and Khosla P., "An Analysis of Manipulator Force Control Strategies Applied to an Experimentally Derived Model", IEEE/RSJ International Conference on Intelligent Robots and Systems, Raleigh, July, 1992.
[50] Volpe R. and Khosla P., "Computational Considerations in the Implementation of Force Control Strategies", Journal of Intelligent and Robotic Systems, 9-1994
[51] Volpe R. and Khosla P., "On the Equivalence of Second Order Impedance Control and Proportional Gain Explicit Force Control", to appear in The International Journal of Robotics Research, 1994.
[52] Siciliano B., "Parallel Force/Position Control of Robot Manipulators", Proceedings of the 7th International Symposium of Robotics Research, Springer-Verlag, London, UK, 1996:79-89.
[53] Chiaverini, S., "Force/Position Regulation of Compliant Robot Manipulators", IEEE Transactions on Automatic Control, Março de 1994.
[54] Pires, JN, "MATJR3PCI", Users Manual of the JR3PCI Matlab Toolbox, http://robotics.dem.uc.pt/norberto/jr3pci/, 2001.
[55] Pires, JN, "Using Matlab to Interface Industrial Robotic & Automation Equipment", IEEE Robotics and Automation Magazine, September 2000.
[56] JR3 Force/Torque Sensor Users Manual, JR3 Inc. Woodland, California, 2001.
[57] JR3 PCI Web Site, http://robotics.dem.uc.pt/norberto/jr3pci/, 2001.

http://www.springer.com/978-0-387-23325-3

Industrial Robots Programming
Building Applications for the Factories of the Future
Pires, J.N.
2007, XIV, 282p. 125 illus., Hardcover
ISBN: 978-0-387-23325-3